CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS
AVANZADOS DEL INSTITUTO POLITÉCNICO NACIONAL
DEPARTAMENTO DE COMPUTACIÓN

Optimización multiobjetivo mediante un algoritmo híbrido basado en cómputo evolutivo y métodos clásicos de optimización

Tesis que presenta

Saúl Zapotecas Martínez

Para obtener el grado de

Maestro en Ciencias

en la especialidad de

Ingeniería Eléctrica

Director de la Tesis: **Dr. Carlos A. Coello Coello**

México, D. F. Diciembre 2007

Dedicatoria

Con todo mi amor a mis padres Lauro e Inés, quienes con sus esfuerzos, desvelos y preocupaciones, me han proveído de todo lo necesario para lograr mis metas.

A mi hermana Vero, a quien quiero mucho y a quien brindo todo mi apoyo en lo que en un futuro pudiera requerir.

A la memoria de mis abuelos Marina, Cirilo y Andrea.
Sin olvidar a mi tía Natalia. Descansen en paz.

AGRADECIMIENTOS

A mis padres y a mi herma por el apoyo que me brindaron durante toda esta ventura. Sin ustedes, mis logros no serían posibles.

A mis primos Victor y Maru, quienes me brindaron su apoyo incondicional durante este proyecto. Y no olvidar al pequeño Diego, quién inconcientemente me mantuvo despierto en momentos necesarios.

A todos mis compañeros de generación. En especial a Cuahutemoc, Jorge y Daniel con quienes compartí desvelos y buenos momentos. No olvidar también a Marco, Victor, William, Carlos y Juanito, gracias por sus comentarios acertados y no tanto. Fue un gusto conocerlos.

A Sofi quién nos ofrece todo su apoyo incondicional, gracias por preocuparte y ocuparte de nosotros.

Al Dr. Carlos A. Coello, por haberme dado la oportunidad de elaborar la tesis bajo su dirección y por el apoyo incondicional brindado, en momentos difíciles durante la maestría.

Al Dr. Luis Gerardo de la Fraga y a la Dra. Xiaoou Li, por los comentarios acertados a esta tesis y lograr con ello, un mejor trabajo.

Al CINVESTAV por permitir formar parte de esta gran institución.

Al CONACyT por el apoyo económico brindado, sin éste no hubiese podido llevar a cabo este proyecto.

Este trabajo de tesis se derivó del proyecto CONACyT titulado “Artificial Immune Systems for Multiobjective Optimization” (Ref. 42435-Y), cuyo responsable es el Dr. Carlos A. Coello Coello.

RESUMEN

En el mundo real, existe una gran cantidad de problemas de optimización para los cuales los métodos clásicos de programación matemática no pueden garantizar que la solución obtenida sea óptima. Además, estos métodos pueden resultar poco eficientes y en algunos casos inoperables para un determinado problema. Para estos problemas más complejos de optimización, es cuando se justifica plenamente el uso de metaheurísticas. Los algoritmos evolutivos son metaheurísticas que en los últimos años se han vuelto muy populares debido a su simplicidad conceptual y su eficiencia en este tipo de problemas.

Los algoritmos evolutivos han sido utilizados exitosamente en el área de optimización multiobjetivo, debido a que por su naturaleza (basada en poblaciones) permiten generar varias soluciones del conjunto de óptimos de Pareto en una sola ejecución. Desafortunadamente, cuando la función objetivo es computacionalmente costosa de evaluar, los algoritmos evolutivos suelen volverse imprácticos.

En esta tesis presentamos dos nuevos algoritmos híbridos basados en cómputo evolutivo y métodos clásicos de optimización. El primero denominado *Nonlinear Simplex Search Differential Evolution* (NSSDE) para optimización global, está basado en la heurística de *evolución diferencial* (ED) y utiliza el método de Nelder-Mead como buscador local. La finalidad de esta implementación es diseñar una estrategia de búsqueda local, que pueda ser utilizada de manera similar en el problema de optimización multiobjetivo (el cual es el objetivo principal de esta tesis). El segundo algoritmo híbrido denominado *Nonlinear Simplex Search Genetic Algorithm* (NSS-GA) para optimización multiobjetivo, está basado en el algoritmo evolutivo multiobjetivo *Non-dominated Sorting Genetic Algorithm-II* (NSGA-II), acoplando los métodos clásicos de optimización matemática de Nelder-Mead (para funciones multidimensionales) y el de la sección dorada (para funciones unidimensionales) que fungen como buscadores locales.

La motivación principal de estas dos hibridizaciones, es aprovechar el poder explorativo de los algoritmos evolutivo y el poder explotativo de los métodos de programación matemática. Los resultados obtenidos en nuestras dos propuestas NSSDE y NSS-GA, son comparados con respecto a los obtenidos por los algoritmos evolutivos simples ED y NSGA-II, respectivamente. Nuestras propuestas algorítmicas muestran tener mejores resultados, en la mayoría de las funciones de prueba adoptadas.

ABSTRACT

In the real world, there are a lot of optimization problems for which traditional methods of mathematical programming cannot guarantee that the solution obtained is optimun. Furthermore, these methods can be inefficient and in some times inoperable for a particular problem. For these more complex optimization problems, the use of metaheuristics is fully justified. The evolutionary algorithms are metaheuristics which in the recent years have become very popular because for their conceptual simplicity and efficiently in these types of problems.

Evolutionary algorithms have been used successfully in the multiobjective optimization area, for their nature (based on a population) allow to generate multiple solutions of the Pareto optimal set in a single run. Unfortunately, when the function is computationally expensive to evaluate, evolutionary algorithms often become impractical.

In this thesis, we present two new hybrid algorithms based on evolutionary computation and classical optimization methods. The first algorithm called *Nonlinear Simplex Search Differential Evolution* (NSSDE) for global optimization, is based on the *Differential Evolution* heuristic and uses the Nelder-Mead method as its local search mechanism. The purpose of this implementation, is to design a local search strategy, which can be used in an analogous way in the multiobjective optimization problem (which is the main objective of this thesis). The second hybrid algorithm proposed is called *Nonlinear Simplex Search Genetic Algorithm* (NSS-GA) for multiobjective optimization. This algorithm is based on the *Non-dominated Sorting Genetic Algorithm-II* (NSGA-II), coupled with the classical methods of mathematical programming Nelder-Mead (for multidimensinal functions) and the golden section (for unidimensional functions) both act as local search engines.

The main motivation for these two hybridizations, is to exploit the explorative power of the evolutionary algorithms and the exploitative power of the mathematical programming methods. The results obtained in our two proposals, NSSDE and NSS-GA, are compared with respect to the results obtained by the original evolutionary algorithms adopted, DE and NSGA-II, respectively. Our proposed approaches have shown better results in most of the test funcions adopted.

ÍNDICE GENERAL

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

ÍNDICE DE ALGORITMOS

CAPÍTULO 1

INTRODUCCIÓN

La optimización es una rama de las matemáticas que busca minimizar o maximizar el valor de una función, eligiendo sistemáticamente los valores de las variables de decisión, en un espacio de búsqueda determinado. En el caso de problemas de optimización con multimples objetivos, se intenta optimizar simultáneamente un conjunto de funciones. Comúnmente, dichas funciones entran en conflicto unas con otras, de tal forma que optimizar una función implica deteriorar el desempeño de otras. A diferencia de los problemas de optimización global en los que sólo existe un único valor óptimo, en los problemas multiobjetivo se trata de encontrar un compromiso entre las distintas funciones objetivo, por lo que se suele obtener un conjunto de soluciones. De esta manera, en un problema de optimización multiobjetivo se puede presentar un conjunto incontable de soluciones, las cuales, cuando son evaluadas, producen vectores cuyos componentes representan los compromisos (*trade-offs*) en el espacio de las funciones objetivo [10].

1.1. Antecedentes y motivación

Muchos problemas de optimización han logrado resolverse satisfactoriamente mediante métodos de programación matemática. Sin embargo, a la fecha no existe ningún método determinístico que pueda resolver todos los problemas no lineales de optimización (ya sea mono-objetivo o multiobjetivo) en su caso más general, garantizando optimalidad en las soluciones obtenidas. Para el caso particular de los problemas no lineales más complejos (p. ej., con espacios de búsqueda muy grandes, accidentados, con alta multimodalidad, etc.), es cuando se justifica más plenamente el uso de metaheurísticas (como el cómputo evolutivo) como un método alternativo de optimización.

La motivación principal de utilizar algoritmos evolutivos (AE) para resolver problemas de optimización multiobjetivo, se debe principalmente a su inherente paralelismo y a su capacidad para explotar el vecindario de las soluciones, mediante el operador de recombinación. La ventaja que tienen los algoritmos evolutivos en estos problemas, radica

en la población con la que trabajan. Esto permite al algoritmo, generar varios elementos del conjunto de óptimos de Pareto en una sola ejecución. Por otro lado, la complejidad que tienen los problemas de optimización multiobjetivo, pueden volver ineficaces (o incluso inoperantes) a las técnicas tradicionales desarrolladas para este tipo de problemas.

Un híbrido se define como "*el producto de combinar dos elementos de distinta naturaleza*" [60].

En este trabajo, se combinan conceptos tomados tanto del cómputo evolutivo como de métodos clásicos de optimización. La motivación principal radica en idear un algoritmo híbrido que combine adecuadamente el poder explorativo del algoritmo evolutivo, con el poder de explotación de un método de programación matemática. Este esquema es similar a la estrategia utilizada por los algoritmos meméticos [51, 52].

Cómputo evolutivo

La computación evolutiva es una área de investigación en las ciencias de la computación; como su nombre los sugiere, es un tipo de computación que engloba una serie de técnicas inspiradas en la teoría Neo-Darwiniana y es utilizada para resolver problemas de una forma muy particular, con base en el *ensayo-y-error*[1] [21].

La idea de solucionar problemas basándose en los principios de la evolución natural no es nueva. La evolución natural fue vista como un proceso de aprendizaje desde los 1930s. En 1932, Walter D. Cannon en su libro "*The Wisdom of the Body*" plantea que el proceso evolutivo es algo similar al aprendizaje por *ensayo-y-error*, que suele manifestarse en los humanos [6]. En 1950, Alan M. Turing reconoció una conexión "obvia" entre la evolución y el aprendizaje de máquina en su artículo titulado "*Computing Machinery and Intelligence*" [76]. También en 1957, George E. P. Box propuso un método con un enfoque evolutivo llamado "*Evolutionary Operation*" [5] para resolver problemas de optimización industrial.

Durante la década de los 1960s, se desarrollaron tres técnicas inspiradas en la teoría Neo-Darwiniana, en lugares distintos y con motivaciones distintas. Hoy en día, estas tres técnicas evolutivas (*programación evolutiva, estrategias evolutivas* y *algoritmos genéticos*) son consideradas como los principales paradigmas del cómputo evolutivo.

1. *Programación evolutiva*. La programación evolutiva (PE) fue desarrollada por Lawrence J. Fogel a mediados de los 1960s. Fogel simuló la evolución natural como un proceso de aprendizaje, teniendo como objetivo generar inteligencia artificial [23, 22]. Este algoritmo evolutivo está inspirado en el principio de evolución a nivel de las especies. Utiliza una selección probabilística y no requiere cruza o algún tipo de recombinación, puesto que una especie no puede mezclarse con otra.

[1] También conocido como *generar-y-probar*.

2. *Estrategias evolutivas*. Las estrategias evolutivas (EEs) fueron propuestas de forma independiente en 1965, por Ingo Rechenberg [61] y Hans-Paul Schwefel [67] en Alemania. Fueron desarrolladas originalmente para resolver problemas de optimización hidrodinámicos de alto grado de complejidad. En las EEs no sólo se evoluciona a las variables del problema, sino también a los parámetros de la misma técnica (a esto se le denomina *auto-adaptación*). En esta técnica se simula el proceso evolutivo al nivel de los individuos, por lo que la recombinación es posible y se utiliza un mecanismo de selección determinístico. La propuesta original de la EE no contaba con población, pero más adelante fue introducida por Schwefel [68].

3. *Algoritmos genéticos*. Los algoritmos genéticos (AGs), originalmente denominados "planes reproductivos genéticos" [30], fueron desarrollados a principios de los 1960s por John H. Holland, con el objetivo de resolver problemas de aprendizaje de máquina. Se caracterizan principalmente por la codificación de los individuos (tradicionalmente se usa una cadena binaria); el mecanismo de selección (probabilístico); simulan la evolución a nivel de individuos, siendo la cruza sexual el operador principal y la mutación un operador secundario.

   Los AGs trabajan a nivel genotípico y no utilizan un mecanismo de auto-adaptación como las EEs, aunque el uso de dicha estrategia es posible y ha sido explorado en algunos trabajos del área [13, 65]. Por otra parte, el elitismo[2] juega un papel crucial en los AGs. Rudolph [63] demostró que un AG requiere elitismo para poder converger al óptimo. Es por esto, que el elitismo es un mecanismo que se ha vuelto estándar en los AEs modernos.

1.2. Planteamiento del problema

Las ténicas clásicas de programación matemática han sido desarrolladas y probados desde hace tiempo, ofreciendo buenos resultados para determinados problemas. Desafortunadamente, no siempre es posible aplicar estos métodos matemáticos. El cómputo evolutivo ha sido utilizado para resolver problemas de optimización más complejos y en la mayoría de casos ha dado mejores resultados que los métodos tradicionales de optimización. Por su naturaleza, los algoritmos evolutivos están basados en poblaciones y por tal motivo, requieren un número de evaluaciones de la función objetivo en el proceso de optimización que puede resultar elevado en ciertas aplicaciones.

Cuando dicha función es computacionalmente cara de evaluar, los algoritmos evolutivos se vuelven imprácticos. Reducir el número de evaluaciones de la función de aptitud, en un algoritmo evolutivo multiobjetivo del estado del arte, empleando métodos de programación matemática como buscadores locales, es precisamente el tema que se aborda en esta tesis. Para validar la eficiencia del algoritmo híbrido propuesto, se le usará para optimizar un

[2] Se denomina *elitismo* al hecho de retener intacto al mejor individuo de cada generación.

conjunto de funciones que en la literatura han sido catalogadas como difíciles para un algoritmo evolutivo multiobjetivo moderno.

1.3. Objetivos

General

El objetivo general de esta tesis, es diseñar un algoritmo multiobjetivo híbrido que combine un algoritmo evolutivo con una técnica de programación matemática y que reduzca el número de evaluaciones de la función de aptitud. El algoritmo evolutivo multiobjetivo adoptado como base para la hibridización es el *Non-dominated Sorting Genetic Algorithm-II* (NSGA-II), al cual se le acopla el algoritmo de Nelder-Mead (para funciones multidimensionales) y el de la sección dorada (para funciones unidimensionales) como métodos de búsqueda local. Estos dos algoritmos son técnicas de programación matemática.

Particulares

- Diseñar un algoritmo híbrido mono-objetivo, con la finalidad de desarrollar una estrategia de búsqueda local que pueda ser utilizada en el caso de optimización multiobjetivo.
- Diseñar una estrategia para la cual se puedan resolver problemas de optimización multiobjetivo, empleando métodos tradicionales de optimización global.
- Mostrar que la implementación del buscador local mejora considerablemente el desempeño del algoritmo evolutivo multiobjetivo adoptado en la hibridización (en este caso, el NSGA-II).

1.4. Contribuciones

Las principales contribuciones que se hacen en este trabajo son:

- Un algoritmo híbrido mono-objetivo para optimización global basado en la heurística de evolución diferencial y el método de optimización global de Nelder-Mead.
- Un algoritmo híbrido multiobjetivo basado en el NSGA-II, acoplando los métodos clásicos de optimización de Nelder-Mead (para funciones multidimensionales) y el de la sección dorada (para funciones unidimensionales), que son utilizados como buscadores locales.
- El diseño de una estrategia de búsqueda local que es implementada en los algoritmos híbridos, la cual da mejores resultados en comparación con los algoritmos evoltivos simples adoptados.

1.5. Organización de la tesis

El resto de la tesis se describe a continuación:

Capítulo 2. Optimización: Se exponen los conceptos fundamentales de optimización así como los métodos clásicos adoptados para este trabajo. También se aborda el tema de optimización multiobjetivo y se presentan algunas de las técnicas tradicionales para la resolución del mismo.

Capítulo 3. Cómputo evolutivo multiobjetivo: Se presentan algunas de las técnicas evolutivas multiobjetivo que son consideradas como el estado del arte actual, entre ellas, el algoritmo evolutivo multiobjetivo adoptado para este trabajo.

Capítulo 4. Propuesta mono-objetivo: En este capítulo se describe el algoritmo evolutivo mono-objetivo adoptado y se detalla el híbrido mono-objetivo propuesto. También, se presentan los resultados obtenidos al comparar el híbrido mono-objetivo con el algoritmo evolutivo adoptado en su hibridización.

Capítulo 5. Propuesta multiobjetivo: En este capítulo se presenta una descripción detallada del algoritmo híbrido multiobjetivo propuesto. La estrategia usada para acoplar los métodos clásicos de optimización como buscadores locales en el caso mono-objetivo, se retoma para el caso multiobjetivo complementando sus deficiencias. También se presentan los resultados comparativos entre nuestra propuesta y el algoritmo evolutivo multiobjetivo simple adoptado.

Capítulo 6. Conclusiones y trabajo a futuro: El último capítulo de esta tesis, donde se presentan las conclusiones a las que se llegaron con base en los estudios y resultados obtenidos. Además, se exponen las posibles líneas de investigación a seguir a futuro, para extender este trabajo de tesis.

Apéndice A. Funciones de prueba (mono-objetivo): Se presenta el conjunto de funciones utilizadas para las pruebas del híbrido mono-objetivo.

Apéndice B. Funciones de prueba (multiobjetivo): Se presentan los problemas de optimización multiobjetivo utilizados para las pruebas y comparaciones del híbrido multiobjetivo.

Apéndice C. Convergencia en los problemas multiobjetivo: Se presentan las gráficas donde se muestra la convergencia del algoritmo multiobjetivo propuesto, en los problemas multiobjetivo adoptados.

CAPÍTULO 2

OPTIMIZACIÓN

2.1. Antecedentes históricos

La *optimización*, se puede definir como el proceso por el cual se busca minimizar o maximizar el valor de una función, eligiendo sistemáticamente los valores de las variables de decisión, dentro de un espacio de búsqueda permisible.

La *investigación de operaciones*, es una rama de las matemáticas que engloba a técnicas y métodos científicos aplicables a problemas de toma de decisiones, cuyo objetivo es establecer la mejor solución, es decir, la óptima. Los *métodos de optimización*, son estudiados en el área de investigación de operaciones y su existencia se remonta a la época de Isaac Newton (1643–1727), Joseph-Louis Lagrange (1736–1813) y Augustin-Louis Cauchy (1789–1857).

El desarrollo de los métodos de optimización basados en el cálculo diferencial, fue posible gracias a las aportaciones que Isaac Newton y Gottfried W. von Leibnitz (1646–1716) hicieron a dicha área. Johann Bernoulli (1667–1748), Leonhard Euler (1707–1783), Joseph-Louis Lagrange y Karl Weirstass (1815–1897) sentaron los fundamentos del cálculo de variaciones, que lidia con la minimización de funciones. La primera aplicación del método del *gradiente descendiente* (conocido también como *descenso empinado*), fue desarrollada por Cauchy para resolver problemas de minimización en espacios continuos. Mientras que el método de optimización para problemas en espacios restringidos, fue desarrollado por Lagrange, el cual involucra la adición de multiplicadores conocidos hoy en día como *multiplicadores de Lagrange* en honor a su inventor.

A pesar de estas primeras contribuciones, se logró muy poco avance en el área. Fue hasta mediados del siglo XX, cuando las computadoras digitales de alta velocidad, hicieron posible la implementación de los algoritmos de optimización existentes estimulándose el desarrollo de nuevos métodos. Esto marcó el inicio de una época increíblemente productiva

en la que surgió un volumen considerable de publicaciones sobre optimización. Además, surgieron varias áreas bien definidas dentro de la teoría de la optimización.

Los principales desarrollos en el área de métodos numéricos para la optimización de funciones sin restricciones, se llevaron a cabo en el Reino Unido en la década de los 1960s. El método simplex fue desarrollado por George Dantzing (1914–2005) en 1947 y el principio de optimalidad para problemas de programación dinámica fue anunciado en 1957 por Richard Bellman (1921–1984). Lo anterior, sentó las bases para el desarrollo de métodos de optimización en espacios restringidos. Sin embargo, fue hasta 1951 con el trabajo de Harold W. Kuhn (1925–) y Albert W. Tucker (1905–1995) en torno a las condiciones necesarias y suficientes para la solución de problemas de optimización, que se establecieron como fundamentos para el desarrollo de una gran cantidad de investigación en optimización no lineal. Aunque no ha sido encontrada alguna técnica aplicable a todos los problemas de optimización no lineal, los trabajos de C. W. Carroll, Anthony V. Fiacco y Garth P. McCormick, permitieron resolver una amplia gama de problemas complejos con restricciones utilizando funciones de penalización.

La década de los 1960s se caracterizó por el surgimiento de nuevas técnicas de optimización, con enfoques distintos. La programación geométrica fue desarrollada por R. J. Duffin, C. Zener y E. L. Peterson a principios de los 1960s. Por otro lado, Ralph E. Gomory fue pionero en realizar trabajos sobre programación entera, la cual ha sido una de las áreas de mayor crecimiento en optimización. George Dantzig, Abraham Charnes y William W. Cooper desarrollaron técnicas de programación estocástica y resolvieron problemas en los cuales, se presupone que los parámetros de diseño son independientes y con una distribución normal.

El deseo de optimizar simultáneamente más de una función objetivo, motivó el desarrollo de métodos de programación multiobjetivo. Uno de los primeros métodos multiobjetivo fue la programación de metas (*goal programming*), propuesta por Charnes y Cooper en 1961 para problemas lineales. Los fundamentos de la teoría de juegos (la cual guarda también relación con la optimización multiobjetivo) fueron estudiados en 1928 por John von Neumman y desde entonces, esta técnica ha sido aplicada para resolver varios problemas de economía.

La segunda mitad del siglo XX se caracterizó por el advenimiento de técnicas heurísticas, tales como: el recocido simulado, los algoritmos genéticos y las redes neuronales. Esta clase de métodos, han dado mejores resultados para los problemas de optimización más complejos (p.ej., en espacios accidentados, funciones altamente no lineales, etc.).

Los problemas de optimización pueden clasificarse de acuerdo a distintos criterios. El que concierte a este trabajo, es en torno al número de funciones objetivo. En las siguientes secciones presentaremos algunos conceptos básicos y fundamentales en el desarrollo de esta tesis, en su mayoría tomados de [1] y [10].

2.2. Optimización global

La optimización es el proceso de encontrar un valor mínimo o máximo global de una función, en un espacio de búsqueda permitido. El problema de optimización global, puede ser definido como:

DEFINICIÓN 1 (MÍNIMO GLOBAL)
Dada una función $f : \Omega \subseteq \mathbb{R}^n \mapsto \mathbb{R}$, $\Omega \neq \phi$, para $\vec{x}^* \in \Omega$ el valor $f^* = f(\vec{x}^*) > -\infty$ es llamado *mínimo global*, si y sólo si:

$$\forall \vec{x} \in \Omega : f(\vec{x}^*) \leq f(\vec{x}) \tag{2.1}$$

De esta manera, el vector $\vec{x}^*$ es un *punto mínimo global*, f es la función objetivo, y el conjunto Ω es el subespacio de soluciones factibles. El problema de determinar un punto mínimo global es llamado el *problema de optimización global* y en este caso, se tendrá un único valor óptimo global.

DEFINICIÓN 2 (RESTRICCIONES)
Sea $\Omega = \{\vec{x} \in \mathbb{R}^n | g_i(\vec{x}) \geq 0 \forall i \in \{1,2,\ldots,q\}\}$ la región factible de la función objetivo $f : \Omega \to \mathbb{R}$. Las funciones $g_i : \mathbb{R}^n \to \mathbb{R}$ son llamadas *restricciones*, y en el punto $\vec{x} \in \mathbb{R}^n$ es llamada una *restricción*:

$$\begin{array}{lcll} \text{satisfecha} & \Leftrightarrow & g_j(\vec{x}) \geq 0, & \\ \text{activa} & \Leftrightarrow & g_j(\vec{x}) = 0, & \\ \text{inactiva} & \Leftrightarrow & g_j(\vec{x}) > 0, & \text{y} \\ \text{violada} & \Leftrightarrow & g_j(\vec{x}) < 0. & \end{array}$$

El problema de optimización global es llamado *sin restricciones*, si y sólo si, $\Omega = \mathbb{R}^n$; en otro caso, es llamado *con restricciones*.

DEFINICIÓN 3 (MÍNIMO LOCAL)
Para $\hat{\vec{x}} \in \Omega$ el valor $\hat{f} = f(\hat{\vec{x}})$ es llamado mínimo local, si y sólo si:

$$\exists \varepsilon \in \mathbb{R}, \varepsilon > 0 : \forall \vec{x} \in \Omega : |\vec{x} - \hat{\vec{x}}| < \varepsilon \Rightarrow \hat{f} \leq f(\vec{x}) \tag{2.2}$$

En otras palabras, un *ambiente*-ε $U_\varepsilon(\hat{\vec{x}}) = \{\vec{x} \in \Omega : |\vec{x} - \hat{\vec{x}}| < \varepsilon\}$ existe, tal que $\hat{f}$ es el valor factible más pequeño de la función objetivo, dentro de ese ambiente (ver figura 2.1).

El problema de maximización puede reducirse a un problema de minimización, si la función a maximizar se multiplica por -1. En estas condiciones, el valor del máximo de la función será igual a menos el mínimo del negativo de la función, es decir:

$$\text{máx}\{f(\vec{x})\} = -\min_{\forall \vec{x} \in \Omega} \{-f(\vec{x})\}$$

Debido al evidente isomorfismo existente entre los problemas de minimización y maximización, de ahora en adelante, hablaremos de optimización refiriéndonos al problema de minimización.

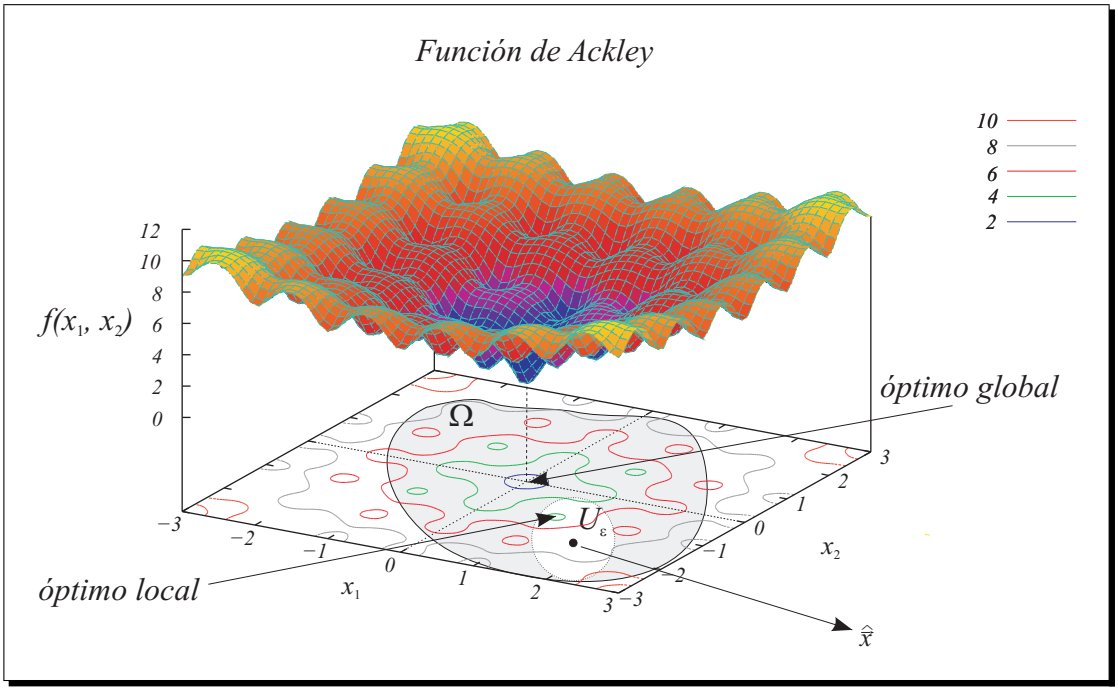

Figura 2.1: Mínimo global y mínimo local

2.3. Métodos de optimización mono-objetivo

En los antecedentes históricos mencionamos una serie de métodos de optimización que fueron desarrollados a lo largo de los años. Lo que ha motivado el desarrollo de dichos algoritmos, han sido los diferentes problemas que existen en el mundo real. Por esta razón, se han propuesto distintas taxonomías de estos métodos [59, 14]. Por cuestiones de simplicidad, nosotros clasificamos a estos métodos en dos categorías: *clásicos* y *no clásicos* (o estocásticos, ver figura 2.2).

A continuación describiremos brevemente estas dos categorías y detallaremos los métodos clásicos utilizados en este trabajo.

2.3.1. Métodos clásicos

Los métodos clásicos de optimización son algoritmos determinísticos que se caracterizan por tener reglas específicas para moverse entre una solución y otra. Estos métodos han sido utilizados desde hace tiempo y han sido aplicados exitosamente en muchos problemas de diseño, en el área de ingeniería. En la actualidad, existe una gran variedad de estos métodos. Puede encontrarse una amplia recopilación de ellos en [59] y [14]. En estos métodos, se distinguen dos grandes grupos: los métodos *basados en derivadas* y los métodos de *búsqueda directa*, que han sido definidos de acuerdo a sus bases conceptuales (ver figura 2.2).

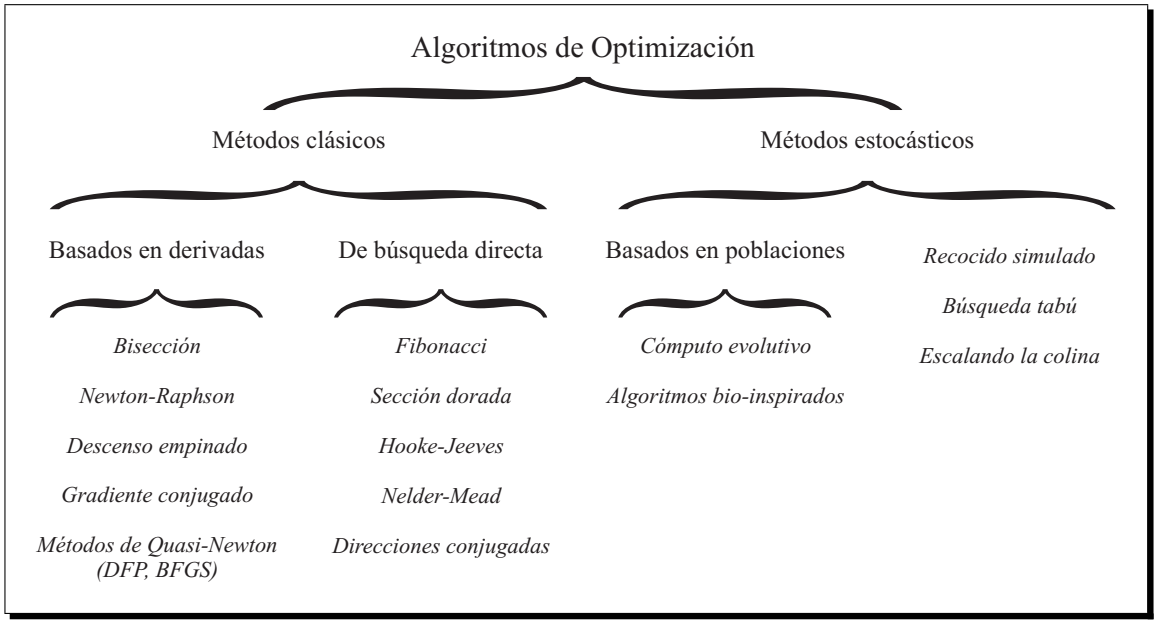

Figura 2.2: Una taxonomía de los algoritmos de optimización

i. Métodos basados en derivadas.

Estos métodos utilizan información de la derivada de la función, como estrategia para moverse entre una solución y otra. En la literatura de investigación de operaciones, existe un gran número de estos algoritmos, que han sido aplicados tanto a problemas unidimensionales como a problemas multi-dimensionales de optimización. De esta manera, los método de bisección, secante y Newton-Raphson, son utilizados para resolver problemas unidimensionales. Mientras que el gradiente conjugado de Fletcher-Reeves, el gradiente descendente de Cauchy y los métodos de Quasi-Newton (p.ej. el método de Davidon-Fletcher-Powell (DFP) y el método de Broyden-Fletcher-Goldfarb-Shanno (BFGS)), fueron diseñados para tratar problemas multi-dimensionales.

ii. Métodos de búsqueda directa.

Este tipo de métodos no necesitan ninguna información de la derivada de la función y son una alternativa complementaria de los métodos anteriores. De igual manera, existen algoritmos para solucionar problemas unidimensionales como los métodos de la sección dorada, Fibonacci y métodos para resolver problemas multi-dimensionales como el algoritmo de Hooke-Jeeves, Nelder-Mead y las direcciones conjugadas de Powell, entre otros muchos más.

Desafortunadamente, estos métodos no garantizan converger al óptimo global. En la mayoría de los casos, estos métodos dependen de un punto inicial de búsqueda y en funciones multi-modales es común quedar estancados en regiones donde existen óptimos locales.

A continuación describiremos los *métodos clásicos* que han sido adoptados en nuestra propuesta algorítmica.

2.3.1.1. Método de la sección dorada

El método de la sección dorada localiza mínimos a partir de un rango inicial y es eficiente para optimizar funciones sin restricciones, unimodales y unidimensionales. El método está basado en el principio de eliminación de regiones, el cual establece que:

Dado dos puntos $x_1, x_2 \in (a,b)$, tal que, $x_1 < x_2$. Para funciones unimodales de minimización, se concluye que:

i . Si $f(x_1) > f(x_2)$ entonces el mínimo no está en el intervalo (a, x_1).

ii . Si $f(x_1) < f(x_2)$ entonces el mínimo no está en el intervalo (x_2, b).

iii . Si $f(x_1) = f(x_2)$ entonces el mínimo no está en el intervalo (a, x_1) y (x_2, b).

La idea general de este principio es eliminar regiones en donde no se encuentre el mínimo de la función, basándose en la *sección dorada*.

Se le conoce como sección dorada, al segmento de recta dividido en dos partes de acuerdo a la *proporción áurea*, la cual establece: "*La longitud total del segmento $a+b$ es a la del segmento a, como la magnitud del segmento a es a la de b*" (figura 2.3).

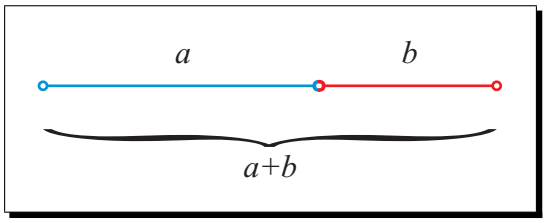

Figura 2.3: Proporción áurea $\frac{a}{b} = \frac{a+b}{a}$

En estas condiciones, la razón *áurea* o *divina* denotada por φ se deduce a partir de la relación anterior, como sigue:

$$\begin{aligned}
\frac{a}{b} = \frac{a+b}{a} \quad &\Longleftrightarrow \quad a^2 = b(a+b) \\
&\Longleftrightarrow \quad a^2 - ab - b^2 = 0 \\
&\Longleftrightarrow \quad a = \frac{b \pm \sqrt{5b^2}}{2} \\
&\Longleftrightarrow \quad a = b\left(\frac{1 \pm \sqrt{5}}{2}\right) \\
&\Longleftrightarrow \quad \frac{a}{b} = \frac{1 \pm \sqrt{5}}{2}
\end{aligned}$$

dado que $a > 0$ y $b > 0$, entonces: $\varphi = \frac{a}{b} = \frac{1+\sqrt{5}}{2} \approx 1.618033$. Y su reciproco comúnmente denotado por Φ, es igual a: $\varphi^{-1} = \frac{b}{a} \approx 0.618033$.

El primer paso de este método, consiste en normalizar la variable de la función x y sus límites (a,b). Al normalizar el espacio de búsqueda se mapea a un intervalo $(0,1)$. De esta forma, en la primera iteración se evalúa 2 veces la función objetivo. Después de aplicar el principio de eliminación de región sólo se evalúa una vez debido a que el intervalo de búsqueda se actualiza en un punto. El método de optimización de la sección dorada reduce entonces el espacio de búsqueda en un Φ^{n-1}, después de n evaluaciones de la función.

En el algoritmo 1, presentamos el pseudo-código del método de optimización de la sección dorada.

Algoritmo 1: Algoritmo de la sección dorada

Input: Intervalo de búsqueda $[a,b]$, una tolerancia ε y un número máximo de iteraciones max_{it}

Output: x^* (punto mínimo encontrado)

```
1  begin
2      a_w = 0, b_w = 1, L_w = b_w − a_w;
3      it = 0;
4      repeat
5          w_1 = a_w + ΦL_w;
6          w_2 = b_w − ΦL_w;
7          x_1 = w_1 ∗ (b − a) + a;
8          x_2 = w_2 ∗ (b − a) + a;
9          if f(x_1) < f(x_2) then
10             a_w = w_2;
11         else
12             b_w = w_1;
13         end
14         L_w = b_w − a_w;
15         it = it + 1;
16     until (|L_w| < ε ó max_it ≤ it) ;
17     x* = (x_1 + x_2)/2;
18 end
```

2.3.1.2. Método de Nelder-Mead

Dentro de las técnicas de optimización de búsqueda directa se encuentra el algoritmo de Nelder-Mead [53] o *simplex no lineal*, propuesto originalmente por Spendley [69] para minimizar funciones multidimensionales, sin restricciones y con dominio en los reales. Este método se basa en la definición de un simplex Δ de $n+1$ vértices, donde n es el espacio euclidiano del dominio de la función. Antes de describir el método, veamos la siguiente definición.

DEFINICIÓN 4 (SIMPLEX)
Un *simplex* o *n-simplex* es la cubierta convexa de un conjunto de $n+1$ puntos no coplanares, es decir, independientes y afines entre sí, en un espacio euclidiano de dimensión n. En otras palabras, un simplex es el análogo en n dimensiones de un triángulo, es decir, un *0-simplex* es un punto; un *1-simplex* es un segmento de una línea; un *2-simplex* es un triángulo; y un *3-simplex* es un tetrahedro (en cada caso, con su interior).

De acuerdo a Nelder y Mead, el simplex puede ser construido a partir de un único vértice Δ_0. De esta manera, los siguientes n vértices del simplex se definen con base en:

$$\Delta_i = \Delta_0 + \delta_1 u_i + \sum_{j=1, j\neq i}^{n} \delta_2 u_j \tag{2.3}$$

donde $i = \{1,2,\ldots,n\}$, $\delta_1 = \frac{\alpha}{n\sqrt{2}}\left(\sqrt{n+1}+n-1\right)$, $\delta_2 = \frac{\alpha}{n\sqrt{2}}\left(\sqrt{n+1}-1\right)$ y u_k es el vector unitario en la k-ésima posición. En el caso particular de $\alpha = 1$ se tendrá un simplex con vértices equidistantes; a éste se lo conoce como simplex regular.

Para definir completamente el algoritmo, se deben especificar cuatro parámetros escalares que controlan los movimientos que se realizan en el simplex: reflexión (α), expansión (γ), contracción (β) y reducción (δ). De acuerdo al artículo original de Nelder-Mead, estos parámetros deben satisfacer:

$$\alpha > 0, \qquad \gamma > 1, \qquad 0 < \beta < 1 \qquad \text{y} \qquad 0 < \delta < 1$$

los parámetros que a Nelder y Mead les dieron buenos resultados y han sido adoptados como estándares son:

$$\alpha = 1, \qquad \beta = \frac{1}{2}, \qquad \gamma = 2 \qquad \text{y} \qquad \delta = \frac{1}{2}$$

aunque éstos pueden variar dependiendo del tipo de problema que se desee resolver.

Teniendo en cuenta un *n-simplex* Δ, cuyos vértices $\Delta_i \in \Delta \subset \mathbb{R}^n; i = \{0,1,\ldots,n\}$ están ordenados de acuerdo a: $f_0 \leq f_1 \leq \ldots \leq f_n$ donde $f_i = f(\Delta_i)$, podemos establecer a Δ_0 como el mejor punto y Δ_n como el peor. En estas condiciones, los movimientos que realiza el método de Nelder-Mead en el *simplex* son:

1. **Reflexión**. Calcula la *reflexión* del peor vértice, i.e. del punto Δ_n (figura 2.4.b):

   $$\vec{x}_r = (1+\alpha)\vec{x}_c - \alpha\Delta_n$$

   donde $\vec{x}_c$ es el *centroide* de los n mejores puntos calculado por: $\vec{x}_c = \frac{1}{n}\sum_{i=0}^{n-1}\Delta_i$

2. **Expansión**. Calcula la *expansión* del punto reflejado (figura 2.4.c):

   $$\vec{x}_e = (1+\alpha\gamma)\vec{x}_c - \alpha\gamma\Delta_n$$

3. **Contracción**. Calcula la *contracción* entre $\vec{x}_c$ y el mejor de Δ_n y $\vec{x}_r$:

   a) Contracción externa (figura 2.4.d).

   $$\vec{x}_{ce} = (1+\alpha\beta)\vec{x}_c - \alpha\beta\Delta_n$$

   b) Contracción interna (figura 2.4.e)

   $$\vec{x}_{ci} = (1-\beta)\vec{x}_c + \beta\Delta_n$$

4. **Reducción**. Se calcula evaluando f de los n puntos $\vec{v}_i = \Delta_0 + \delta(\Delta_i - \Delta_0)$, para $i = \{1,\ldots,n\}$. Los nuevos vértices del simplex en la próxima iteración serán $\Delta_0, \vec{v}_1, \vec{v}_2, \ldots, \vec{v}_n$ figura 2.4.f).

La lógica completa del método se muestra en el algoritmo 2.

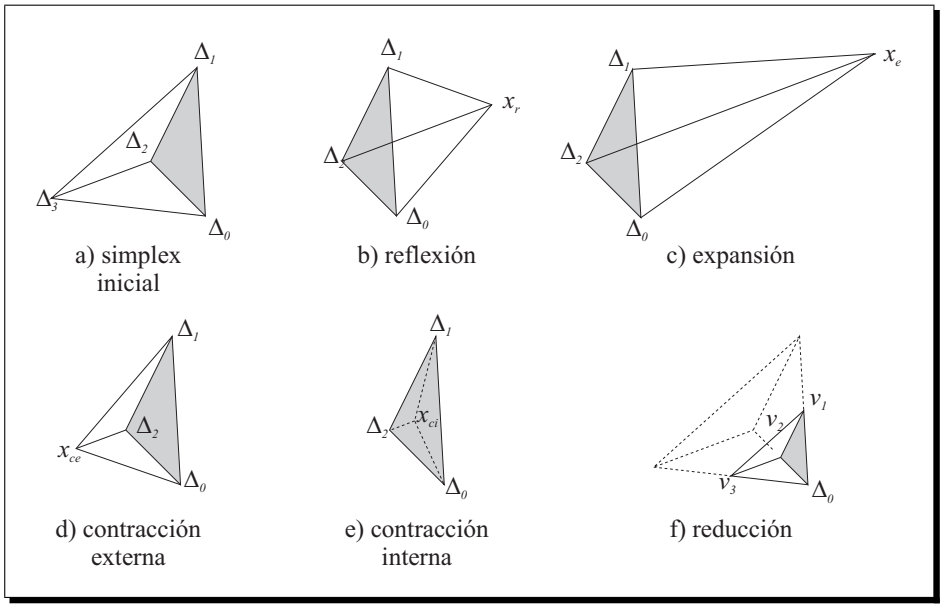

Figura 2.4: Posibles movimientos del algoritmo de Nelder-Mead.
En este caso, el simplex minimiza una función en $\mathbb{R}^3$

Algoritmo 2: Algoritmo de Nelder-Mead

Input: Δ(simplex inicial), f y max_{it}
Output: $\vec{x}^*$ (punto mínimo encontrado)

begin
  $it = 0$;
  while $it < max_{it}$ **do**
    Ordenar ascendentemente cada $\Delta_i \in \Delta$ en base al valor de $f(\Delta_i)$;
    Calcular $\vec{x}_r$; /* reflexión */
    case $f(\Delta_0) \leq f(\vec{x}_r) < f(\Delta_{n-1})$
      $\Delta_n = x_r$;
    end
    case $f(\vec{x}_r) < f(\Delta_0)$
      Calcular x_e; /* expansión */
      if $f(\vec{x}_e) < f(\vec{x}_r)$ **then** $\Delta_n = \vec{x}_e$;
      else $\Delta_n = \vec{x}_r$;
    end
    case $f(\vec{x}_r) \geq f(\Delta_{n-1})$
      if $f(\Delta_{n-1}) \leq f(\vec{x}_r) < f(\Delta_n)$ **then**
        Calcular $\vec{x}_{ce}$; /* contracción externa */
        if $f(\vec{x}_{ce}) \leq f(\vec{x}_r)$ **then** $\Delta_n = \vec{x}_{ce}$;
        else Reducir el simplex Δ; /* reducción */
      else if $f(\vec{x}_r) \geq f(\Delta_n)$ **then**
        Calcular $\vec{x}_{ci}$; /* contracción interna */
        if $f(\vec{x}_{ci}) < f(\Delta_n)$ **then** $\Delta_n = \vec{x}_{ci}$;
        else Reducir el simplex Δ; /* reducción */
      end
    end
    $it++$;
  end
  $\vec{x}^* = \Delta_i | \min_{\forall \Delta_i \in \Delta} \{f(\Delta_i)\}$;
end

2.3.2. Métodos no clásicos

Los métodos no clásicos o estocásticos, se caracterizan por utilizar reglas de transición probabilística. Entre ellos destacan los algoritmos evolutivos, que en los últimos años se han vuelto muy populares. Esta clase de algoritmos en comparación, son nuevos y bastante requeridos debido a que poseen ciertas propiedades de las que los algoritmos determinísticos carecen.

Pero no sólo el cómputo evolutivo se ha empleado para lidiar con problemas de optimización. Algoritmos bio-inspirados como la optimización por cúmulos de partículas (*Particle Swarm Optimization*) [34], colonia de hormigas (*Ant Colony*) [18], además de otras técnicas de búsqueda heurística como el recocido simulado (*Simulated Annealing*) [35] y la búsqueda tabú (*Tabu Search*) [25], han sido de gran utilidad para tratar también con problemas de optimización.

2.4. Optimización multiobjetivo

La optimización multiobjetivo trata de resolver problemas en los cuales se requiere optimizar simultáneamente un conjunto de funciones. Comúnmente, dichas funciones entran en conflicto unas con otras, de tal forma que optimizar una función implica deteriorar el desempeño de otras.

DEFINICIÓN 5 (PROBLEMA DE OPTIMIZACIÓN MULTIOBJETIVO (POM))
Se define como encontrar un vector $\vec{x}^*$ que satisfaga las m restricciones de desigualdad:

$$g_i(\vec{x}^*) \leq 0; i = 1,2,\ldots,m \tag{2.4}$$

las p restricciones de igualdad:

$$h_j(\vec{x}^*) = 0; j = 1,2,\ldots,p \tag{2.5}$$

y optimice el vector de funciones objetivo:

$$\vec{f}(\vec{x}) = [f_1(x), f_2(x), \ldots, f_k(x)]^T$$

El conjunto de restricciones dadas por las ecuaciones 2.4 y 2.5, determina la *región factible* Ω y cualquier vector $\vec{x} \in \Omega$ define una *solución factible*. El vector de funciones $\vec{f}(\vec{x})$ es una función que mapea el conjunto Ω al conjunto Λ y que contiene todos los valores posibles de las funciones objetivo (ver figura 2.5).

DEFINICIÓN 6 (VECTOR IDEAL)
Dado un vector de variables:

$$\vec{x}^{*(i)} = [x_1^{*(i)}, x_2^{*(i)}, \ldots, x_n^{*(i)}]^T$$

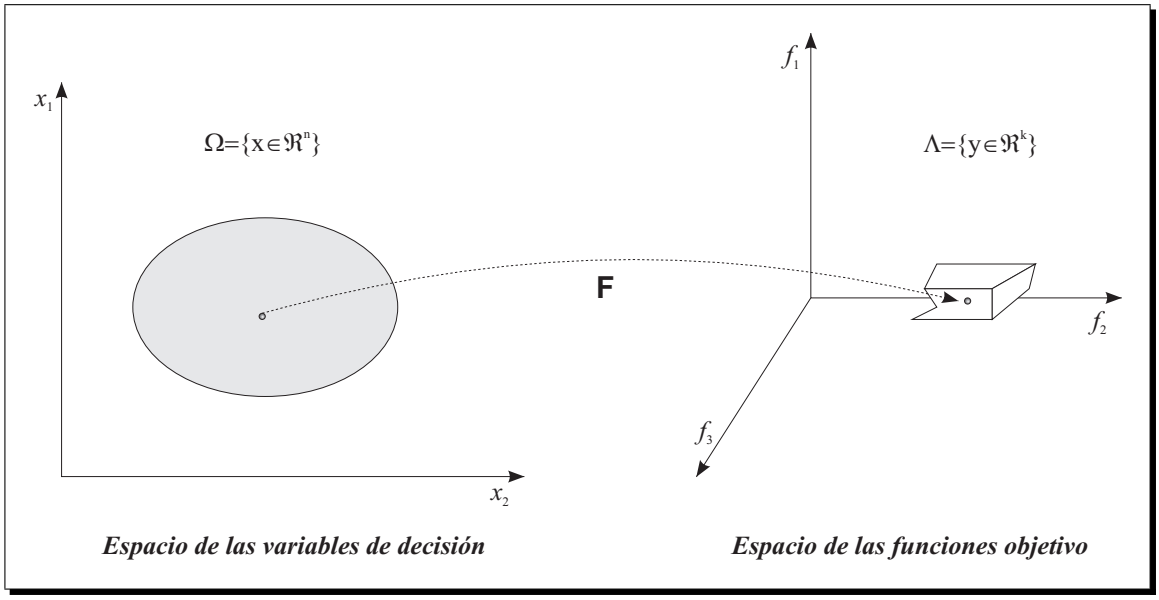

Figura 2.5: Mapeo de evaluación de un POM, para $n = 2$ y $k = 3$

que optimiza la i-ésima función objetivo $f_i(\vec{x})$, es decir, $\vec{x}^{*(i)} \in \Omega$ es tal que:

$$f_i(\vec{x}^{*(i)}) = \textit{óptimo}\, f_i(\vec{x})$$

entonces, el vector $\vec{f}^*(\vec{x}) = [f_1^*(\vec{x}), f_2^*(\vec{x}), \ldots, f_k^*(\vec{x})]^T$ (donde $f_i^*(\vec{x})$ denota el óptimo de la i-ésima función objetivo) es ideal para un POM y el punto correspondiente en $\mathbb{R}^n$ es una solución ideal (utópica).

Cuando se tiene más de una función objetivo, el concepto de óptimo cambia, ya que en este tipo de problemas el objetivo es llegar a un buen compromiso, en lugar de una única solución como en el caso de la optimización global.

La noción de óptimo en problemas multiobjetivo fue originalmente propuesta por Francis Y. Edgeworth en 1881 [20] y generalizada por Vilfredo Pareto en 1896 [55]. El concepto de óptimo de Pareto se presenta formalmente en la siguiente definición.

DEFINICIÓN 7 (ÓPTIMO DE PARETO)
Sea $I = \{1, \ldots, k\}$ y Ω el conjunto de soluciones factibles, decimos que un vector de variables de decisión $\vec{x}^* \in \Omega$ es un óptimo de Pareto, si para toda $\vec{x} \in \Omega$ se cumple:

i. $\forall i \in I : f_i(\vec{x}^*) \leq f_i(\vec{x})$

ii. $\exists j \in I : f_j(\vec{x}^*) < f_j(\vec{x})$

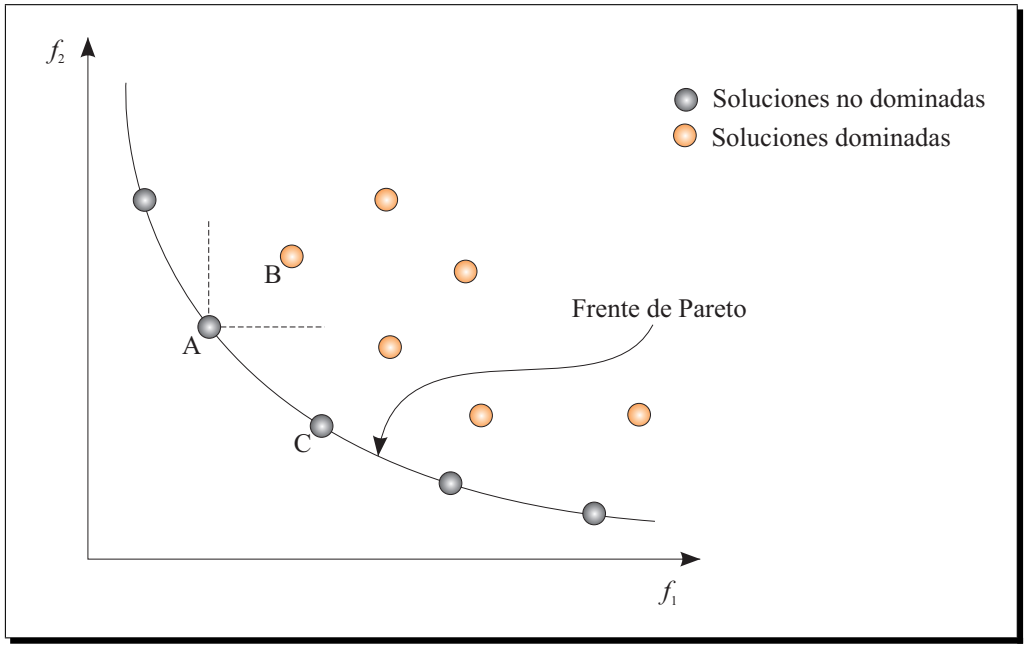

Figura 2.6: Dominancia de Pareto

DEFINICIÓN 8 (DOMINANCIA DE PARETO)
Sea $I = \{1, \ldots, k\}$, se dice que un vector $\vec{u} = (u_1, \ldots, u_k)$ domina a otro $\vec{v} = (v_1, \ldots, v_k)$ (denotado por $\vec{u} \preceq \vec{v}$) si y sólo si $\vec{u}$ es parcialmente menor a $\vec{v}$, es decir, si se cumple:

i. $\forall i \in I : u_i \leq v_i$

ii. $\exists j \in I : u_j < v_j$

En otras palabras, para que una solución domine a otra, ésta debe ser estrictamente mejor en al menos un objetivo y no peor en ninguno de los otros. En la figura 2.6, la solución A domina a la solución B ya que es mejor tanto en f_1 como en f_2, sin embargo, no domina a C ya que no es mejor.

DEFINICIÓN 9 (CONJUNTO DE ÓPTIMOS DE PARETO)
Para un POM dado $\vec{f}(\vec{x})$, el conjunto de óptimos de Pareto denotado por $\mathcal{P}^*$ se define como:

$$\mathcal{P}^* = \{\vec{x} \in \Omega : \nexists \vec{x}^* \in \Omega : \vec{f}(\vec{x}^*) \preceq \vec{f}(\vec{x})\}$$

DEFINICIÓN 10 (FRENTE DE PARETO)
Para un POM dado con un vector de funciones $\vec{f}(\vec{x})$ y un conjunto de óptimos de Pareto $\mathcal{P}^*$, el frente de Pareto, denotado por $\mathcal{PF}^*$, se define como:

$$\mathcal{PF}^* = \{\vec{f}(\vec{x}) = [f_1(\vec{x}), \ldots, f_k(\vec{x})] : \vec{x} \in \mathcal{P}^*\}$$

En general, no es posible tener una expresión analítica de la línea o superficie que contenga a los puntos que conforman el frente de Pareto. Por tanto, el procedimiento normal para generar el frente de Pareto, es obtener el conjunto de puntos factibles Ω y los valores correspondientes de $f(\vec{x})$ para toda $\vec{x} \in \Omega$. Cuando hay un número suficiente de ellos, entonces es posible determinar los puntos no dominados y producir el frente de Pareto.

2.5. Técnicas de optimización multiobjetivo

En esta sección revisaremos brevemente algunas técnicas tradicionales analíticas, que son utilizadas para solucionar POM. Cohon y Marks [11] proponen una clasificación de estas técnicas, la cual ha sido la más popular en la comunidad de investigación de operaciones:

1. *Métodos a priori*. Se toman decisiones antes de la búsqueda (*decidir* $\Rightarrow$ *buscar*). Es decir, el usuario especifica sus preferencias, expectativas y opiniones antes de ejecutar la búsqueda.

2. *Métodos a posteriori*. Buscar antes de tomar decisiones (*buscar* $\Rightarrow$ *decidir*). Después de encontrar el conjunto de óptimos de Pareto, el usuario selecciona las mejores alternativas de acuerdo a su criterio.

3. *Métodos progresivos*. Integra la búsqueda con la toma de decisiones (*decidir* $\Leftrightarrow$ *buscar*). El usuario tiene una participación activa durante el proceso de solución del POM.

Otras clasificaciones son posibles (p.ej., la que presenta Duckstein [19]). Sin embargo, para fines de este trabajo adoptaremos la propuesta hecha por Cohon y Marks, debido a que su clasificación se enfoca a los problemas de búsqueda y toma de decisiones. A continuación presentaremos algunos métodos de acuerdo a la clasificación anterior.

2.5.1. Métodos a priori

2.5.1.1. Método de programación de metas

La programación de metas (*goal programming*) fue desarrollada por Abraham Charnes y William Cooper [8] en 1961, para tratar con POMs en modelos lineales y este trabajo jugó un papel predominante en la aplicación de esta técnica a problemas industriales. En este método, el diseñador tiene asignado sus objetivos o metas que desee lograr para cada objetivo. Estos valores son incorporados al problema como una restricción adicional. La función objetivo trata de minimizar la desviación absoluta entre el valor alcanzado y el objetivo. En su forma más simple este enfoque puede ser formulado como sigue:

$$\text{mín} \sum_{i=1}^{k} |f_i(\vec{x}) - T_i|, \text{ sujeto a } \vec{x} \in \Omega \tag{2.6}$$

donde T_i denota la meta propuesta por el diseñador para la i-ésima función objetivo $f_i(\vec{x})$ y la región factible Ω .

2.5.1.2. Método lexicográfico

En este método, los objetivos se jerarquizan de acuerdo a su importancia en el problema, quedando en primer lugar el más importante. De esta manera, la solución óptima $\vec{x}^*$ es obtenida minimizando las funciones objetivo, empezando con la más importante y procediendo de acuerdo al orden establecido. El valor que se obtiene como consecuencia de cada minimización, se convierte en una restricción para la siguiente función a minimizar y este procedimiento continúa hasta evaluar el último objetivo. El procedimiento de este método se presenta a continuación:

$$\begin{array}{ll} 1. & \min\limits_{x\in\Omega} f_1(\vec{x}) = \alpha_1 \\ 2. & \min\limits_{x\in\Omega}\{ f_2(\vec{x}) | f_1(\vec{x}) \leq \alpha_1\} = \alpha_2 \\ \vdots & \vdots \\ n. & \min\limits_{x\in\Omega}\{ f_n(\vec{x}) | f_1(\vec{x}) \leq \alpha_1, \ldots, f_{n-1}(\vec{x}) \leq \alpha_{n-1}\} = \alpha_n \end{array}$$

En términos generales, la formulación del problema puede escribirse como:

$$\begin{array}{c} \min\limits_{x\in\Omega} f_i(\vec{x}) \\ \text{sujeto a: } \min\limits_{x\in\Omega} f_k(\vec{x}) \leq \alpha_k, \text{ tal que, } k = \{1,2,\ldots,n-1\} \end{array} \tag{2.7}$$

La solución obtenida al final $\vec{x}^*_k$ es tomada como la solución deseada $\vec{x}^*$ del problema.

2.5.2. Métodos a posteriori

2.5.2.1. Combinación lineal de pesos

En 1963, Lofti Zadeh fue el primero en mostrar que con la tercera condición de Kuhn-Tucker sobre soluciones no inferiores, se puede plantear un problema de optimización multiobjetivo como un problema de optimización global, tomando como función objetivo a la suma de las funciones objetivo originales [84]. Lo anterior se puede formular mediante la ecuación 2.8.

$$\begin{array}{c} \min \sum\limits_{i=1}^{k} w_i f_i(\vec{x}) \\ \text{sujeto a: } \vec{x} \in \Omega \end{array} \tag{2.8}$$

donde $w_i \geq 0$ para todo i y es estrictamente positivo para al menos un objetivo. Los pesos w_i no reflejan proporcionalmente la importancia relativa de las funciones objetivo (a menos que los valores de las funciones objetivo estén escalados adecuadamente), pero cuando se varían, se obtienen diferentes soluciones del conjunto de óptimos de Pareto.

2.5.2.2. Método de la restricción ε

El método de la restricción ε, también se deriva de la tercera condición de Kunh-Tucker y ha sido uno de los más utilizados para resolver problemas de optimización multiobjetivo, convexos y no convexos. La idea de este método es minimizar una función objetivo a la vez, considerando los otros objetivos como restricciones por algunos niveles permitidos ε_l. Variando los niveles ε_l, las soluciones no inferiores del problema pueden ser obtenidas. Este problema puede ser formulado como lo muestra la ecuación 2.9.

$$\begin{aligned} &\text{mín } f_r(\vec{x}) \\ &\text{sujeto a: } f_l(\vec{x}) \leq \varepsilon_l \text{ para } l = \{1,2,\ldots,k\} \text{ y } l \neq k \end{aligned} \tag{2.9}$$

donde ε_l asume valores de las funciones objetivo que no deben ser excedidos.

La ventaja principal del método, se presenta cuando se convierte el POM en un problema de optimización global con una restricción ε. En estas circunstancias, es posible utilizar cualquier método de optimización mono-objetivo para minimizar cada función por separado. Para una explicación más detallada del método ver [54].

2.5.3. Métodos progresivos

Los métodos progresivos operan básicamente en tres etapas:

i. Encontrar una solución no dominada.

ii. Se pone a consideración del diseñador la solución encontrada, con el fin de modificar las preferencias de los objetivos de acuerdo a su criterio.

iii. Repetir los pasos *i* y *ii* hasta que el diseñador esté satisfecho de las soluciones obtenidas o no haya mejoras considerables.

A continuación describiremos brevemente uno de estos métodos.

2.5.3.1. Método del escalón

La idea básica es converger hacia la mejor solución en el sentido min-max, en no más de k pasos, donde k es el número de objetivos. Esta técnica es útil, principalmente para resolver problemas lineales. Inicia de un punto ideal y se desarrolla en seis etapas, como plantea Cohon [12].

1. Construir una tabla de soluciones marginales, optimizando cada función objetivo por separado.
2. Calcular, para cada función objetivo:

$$\alpha(i) = \frac{M(i) - m(i)}{M(i)} \left[\sum_{j=1}^{J} c(i,j) \right]^2$$

donde: $M(i) = \text{máx } f_i(\vec{x})$, $m(i) = \text{mín } f_i(\vec{x})$, y $c(i)$ =coeficiente de costo del i-ésimo objetivo lineal. Se asigna cero al índice de la iteración ($k = 0$).

3. Calcular $\prod(i) = \dfrac{\alpha(i)}{\sum \alpha(i)}$ y se resuelve el problema min-max. La solución es llamada $x(k)$.

4. Se presenta la solución al diseñador:

   a) Si la solución es satisfactoria, se detiene el algoritmo.

   b) Si la solución no es satisfactoria y $k < p-1$, ir al paso 5.

   c) Si la solución no es satisfactoria y $k > p-1$, se detiene el algoritmo. Se requiere un procedimiento o al menos una redefinición del problema.

5. El diseñador selecciona un objetivo satisfecho por la solución y determina la cantidad por la cual puede ser decrementada con el fin de mejorar los otros objetivos. Si esto no se puede hacer, entonces se requiere de otro enfoque.

6. Definir una nueva restricción que disminuya el objetivo seleccionado en el paso 5. Sea $\alpha(i) = 0$ para ese objetivo, se incrementa k en uno y se va al paso 3.

La debilidad de esta técnica, radica en que se asume que no existe una mejor solución que $x(k)$, si ésta no es encontrada después de haber ejecutado los seis pasos del algoritmo.

En el siguiente capítulo mencionaremos otro tipo de métodos de optimización multiobjetivo, que están basados en reglas de transición probabilísticas. Concretamente nos referiremos al estado del arte de los algoritmos evolutivos multiobjetivo.

CAPÍTULO 3

CÓMPUTO EVOLUTIVO MULTIOBJETIVO

3.1. Introducción

En la actualidad, existen alrededor de treinta técnicas de optimización multiobjetivo en la literatura de investigación de operaciones [49]. La mayoría de ellas están limitadas a frentes de Pareto con ciertas características (p.ej., convexos). Además, suelen generar una sola solución por ejecución y requerir un punto inicial de búsqueda.

El potencial de los algoritmos evolutivos para resolver problemas de optimización multiobjetivo se remonta a finales de los 1960s, desde la tesis doctoral de Rosenberg [62], la cual indicó la posibilidad de usar algoritmos genéticos en este dominio. Sin embargo, el primer intento real por extender un algoritmo evolutivo a problemas multiobjetivo, es el trabajo desarrollado por J. David Schaffer [64]. Su algoritmo, denominado *Vector Evaluated Genetic Algorithm* (VEGA) fue la entrada formal del cómputo evolutivo a la optimización multiobjetivo.

En cuanto a sus principios básicos, los algoritmos evolutivos multiobjetivo pueden clasificarse en dos grupos. Los que utilizan directamente los conceptos de Pareto y los que no, siendo los primeros los de nuestro interés.

Históricamente, se pueden considerar dos etapas fundamentales que han marcado el desarrollo de los algoritmos evolutivos, basados en los conceptos de Pareto:

1. *Primera generación*. Es una etapa en la cual se desarrollaron algoritmos relativamente simples que hacían uso de enfoques rudimentarios, como funciones agregativas lineales, y métodos lexicográficos. Se desarrollaron también los primeros algoritmos evolutivos multiobjetivo basados en jerarquización de Pareto, los cuales usan primordialmente nichos y compartición de aptitud [16] como su estimador de densidad.

2. *Segunda generación*. Se introduce el concepto de elitismo. De esta manera, se introduce el uso de la selección extintiva o positiva (+) y se populariza el uso de poblaciones secundarias. Los algoritmos de segunda generación enfatizan la eficiencia computacional.

En las siguientes secciones, describiremos brevemente los algoritmos evolutivos multiobjetivos basados en los principios de Pareto, de acuerdo a las etapas de su desarrollo. Pondremos especial énfasis en nuestra discusión del NGSA-II dado que es la técnica adoptada en este trabajo.

3.2. Algoritmos de primera generación

3.2.1. Vector Evaluated Genetic Algorithm (VEGA)

Con el objetivo de superar las dificultades de los métodos agregativos, Schaffer propuso en 1985 el *Vector Evaluated Genetic Algorithm* (VEGA) [64]. En la literatura, VEGA es considerada como una de las técnicas más populares en la categoría de las que no incorporan el concepto de óptimo de Pareto. En términos generales, VEGA divide la población en tantas subpoblaciones como objetivos existan. Estas subpoblaciones se mezclan entre sí a fin de obtener una nueva población del mismo tamaño que la original. En la etapa de selección, cada individuo se elige de acuerdo al objetivo relevante en su población. Después de recombinar a los individuos de la población, se aplican los operadores de cruza y mutación como en un algoritmo genético simple.

El principal problema de VEGA del que Schaffer se percató, es que las soluciones generadas por el algoritmo eran no dominadas localmente, ya que la dominancia se limitaba a la subpoblación en proceso. De esta manera, un individuo que es no dominado en una cierta generación puede resultar dominado por otro individuo que emerja en una generación posterior.

3.2.2. Multi-objective Genetic Algorithm (MOGA)

En 1993, Fonseca y Fleming presentaron una variación de la técnica propuesta por Goldberg [26], a la que llamaron *Multi-objective Genetic Algorithm* (MOGA) [24]. En esta técnica la jerarquía de un individuo corresponde al número de individuos de la población por los cuales es dominado. Consideremos, por ejemplo, un individuo x_i en la generación t, el cual es dominado por $p_i^{(t)}$ individuos en la generación actual. La jerarquía de un individuo es calculada por:

$$rank(x_i,t) = 1 + p_i^{(t)}$$

Todos los individuos no dominados de la población tiene como jerarquía 1, mientras que los dominados, son penalizados de acuerdo a la densidad correspondiente a la región

donde se ubiquen dentro del espacio de las funciones objetivo. El pseudo-código de MOGA se muestra en el algoritmo 3

Algoritmo 3: Multi-objective Genetic Algorithm (MOGA)

Input: Un número máximo de generaciones max_{gen}
Output: Una población evolucionada

```
1  begin
2      Inicializar población;
3      Evaluar población de acuerdo a las funciones objetivo;
4      Asignar jerarquía con base en dominancia de Pareto;
5      Obtener el número de contador de nicho;
6      Asignar aptitud linealmente escalada;
7      Asignar valor de bondad compartido (Shared fitness);
8      for i = 1 to max_gen do
9          Seleccionar mediante muestreo estocástico universal;
10         Cruza en un punto;
11         Mutación;
12         Evaluar población de acuerdo a las funciones objetivo;
13         Asignar jerarquía con base en dominancia de Pareto;
14         Obtener el número de contador de nicho;
15         Asignar aptitud linealmente escalada;
16         Asignar valor de bondad compartido (Shared fitness);
17     end
18 end
```

3.2.3. Niched-Pareto Genetic Algorithm (NPGA)

Este método fue propuesto por Horn y Nafpliotis en 1994 [31]. Consiste en un AG simple, que utiliza una selección mediante torneos binarios basados en la dominancia de Pareto. La idea de la selección consiste en elegir aleatoriamente dos individuos, los cuales son comparados con otros individuos, que están dentro de un subconjunto de la población (comúnmente, 10 % del total de la población). Si uno de los dos individuos domina a los individuos del subconjunto de la población, entonces es el ganador del torneo. Si los dos dominan o son dominados, entonces se aplica el número de contador de nicho y gana el individuo que esté en un nicho con menor densidad. El algoritmo 4 muestra el procedimiento general del NPGA.

3.2.4. Non-dominated Sorting Genetic Algorithm (NSGA)

Esta estrategia fue propuesta por Srinivas y Deb en 1994 [70]. Se basa en clasificar a la población en niveles o capas de dominación (idea original de Goldberg [26]). Los individuos son jerarquizados de acuerdo a su dominancia con otros individuos, de tal manera

Algoritmo 4: Niched-Pareto Genetic Algorithm (NPGA)

Input: Un número máximo de generaciones max_{gen}
Output: Una población evolucionada

```
1  begin
2      Inicializar población;
3      Evaluar población de acuerdo a las funciones objetivo;
4      for i = 1 to max_gen do
5          Aplicar selección de torneo binario especial;
6          case (ind_1 ⪯ ind_2):
7              seleccionar ind_1;
8          case (ind_2 ⪯ ind_1):
9              seleccionar ind_2;
10         case (ind_1 ⪯ ind_2 y ind_2 ⪯ ind_1):
11             seleccionar individuo con el menor contador de nicho;
12         Aplicar cruza en un punto;
13         Aplicar mutación;
14         Evaluar población de acuerdo a las funciones objetivo;
15     end
16 end
```

que los individuos no dominados con respecto a toda la población, pertenecen a la primera capa. A los individuos de la primera capa se les asigna un valor de aptitud “falso” (*dummy fitness*). Para determinar qué individuos pertenecen a las siguientes capas, se eliminan los individuos de la primera capa de la población y se vuelven a determinar las soluciones no dominadas, las cuales pertenecerán a la segunda capa y así sucesivamente con los individuos restantes de la población (ver figura 3.1). Los individuos de la segunda capa tendrán una aptitud “falsa” inferior que la de los individuos de la primera capa y así sucesivamente. Se usa también compartición de aptitud [16], pero en el espacio de las variables de decisión. El algoritmo 5 describe el pseudo-código de este método.

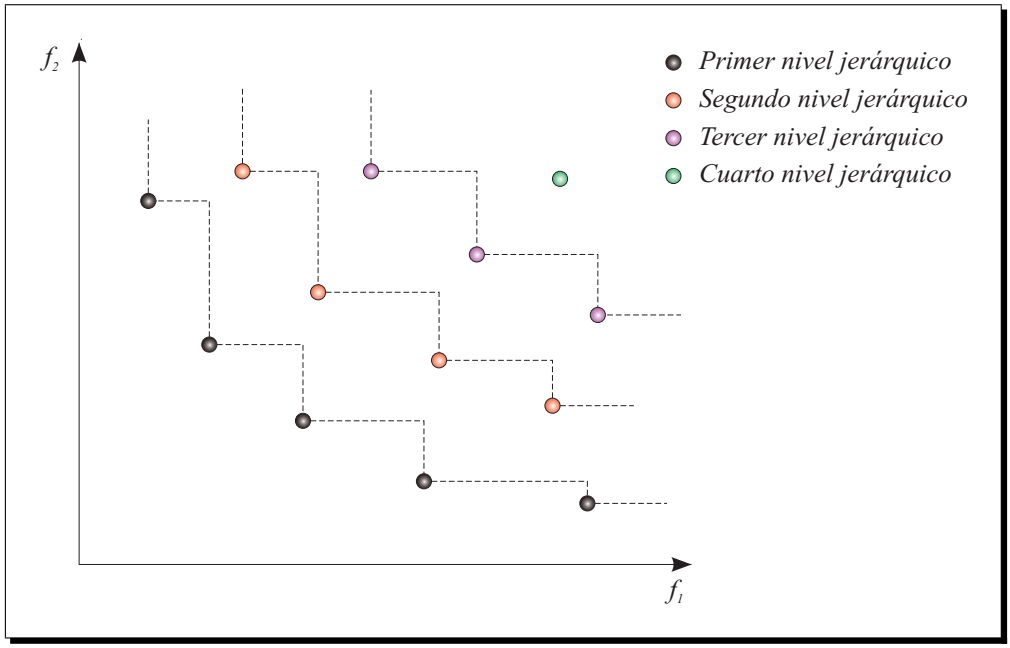

Figura 3.1: Jerarquización de niveles de dominancia

Algoritmo 5: Non-dominated Sorting Genetic Algorithm (NSGA)

Input: Un número máximo de generaciones max_{gen}
Output: Una población evolucionada

```
1  begin
2      Inicializar población;
3      Evaluar población de acuerdo a las funciones objetivo;
4      Asignar jerarquía con base en la dominancia de Pareto en cada frente;
5      Obtener el número de contador de nicho;
6      Se asigna el valor de bondad compartido (Shared fitness);
7      for i = 1 to max_gen do
8          Aplicar selección mediante muestreo estocástico universal;
9          Aplicar operadores de cruza y mutación;
10         Evaluar población de acuerdo a las funciones objetivo;
11         Asignar jerarquía con base en la dominancia de Pareto en cada frente;
12         Obtener el número de contador de nicho;
13         Se asigna el valor de bondad compartido;
14     end
15 end
```

3.3. Algoritmos de segunda generación

3.3.1. Strength Pareto Evolutionary Algorithm (SPEA)

El *Strength Pareto Evolutionary Algorithm* (SPEA) fue propuesto por Zitzler y Thiele en 1999 [88]. Su concepción se fundamenta en la integración de diferentes algoritmos evolutivos multiobjetivo. El pseudo-código del método se presenta en el algoritmo 6. SPEA utiliza un archivo que contiene las soluciones no dominadas encontradas previamente (también se le conoce como conjunto externo no dominado). En cada generación, los individuos no dominados se copian al conjunto externo no dominado. Para cada individuo del conjunto externo, se calcula un valor de fortaleza (*strength value*). Este valor es similar a la jerarquía de MOGA y es proporcional al número de soluciones a las que un individuo domina. La aptitud de cada miembro de la población actual, se calcula de acuerdo al valor de fortaleza de todas las soluciones externas no dominadas que éste domina. Para proveer diversidad a la población, se utiliza una técnica de cúmulos (*clustering*) llamada método de enlace promedio (*average linkage method*) [50], la cual penaliza a las soluciones vecinas más cercanas a la que se está evaluando.

Algoritmo 6: Strength Pareto Evolutionary Algorithm (SPEA)

Input: Un número máximo de generaciones max_{gen}
Output: Una población evolucionada

```
1  begin
2      Inicializar población P;
3      Evaluar población de acuerdo a las funciones objetivo;
4      Crear un conjunto vacío E;
5      for i = 1 to max_gen do
6          Copiar las soluciones no dominadas de P a E;
7          Eliminar los elementos de E que son dominados por otro miembro de E;
8          Podar E (utilizando clustering) cuando la capacidad máxima de E ha sido
           excedida;
9          Calcular aptitud de cada individuo en P y en E;
10         Seleccionar mediante torneo binario con reemplazo para seleccionar
           individuos de P + E (operación de unión) hasta que la piscina de la población
           esté llena;
11         Aplicar cruza y mutación;
12         Evaluar población de acuerdo a las funciones objetivo;
13     end
14 end
```

3.3.2. Strength Pareto Evolutionary Algorithm 2 (SPEA 2)

Fue presentado en 2001 por Zitzler, Laumanns y Thiele [87]. Tiene tres diferencias principales con respecto a su antecesor:

i. Incorpora una estrategia de asignación de aptitud de grano-fino (*fine-grained* fitness), que por cada individuo toma en cuenta el número de individuos a los que domina y el número de individuos por los cuales es dominado.

ii. Utiliza una técnica de estimación del vecino más cercano que guía la búsqueda de una manera más efectiva.

iii. Presenta un método mejorado de truncamiento del archivo, que garantiza la preservación de las soluciones de los extremos del frente de Pareto.

Además, el SPEA 2 utiliza un algoritmo de *clustering* de menor complejidad computacional que su predecesor.

3.3.3. Pareto Archived Evolution Strategy (PAES)

La *Pareto Archived Evolution Strategy* (PAES) fue presentada por Knowles y Corne en el año 2000 [37]. La idea del enfoque es muy simple tal y como se muestra en el pseudo-código del algoritmo 7. PAES consiste en una estrategia evolutiva (1+1), en combinación con un archivo histórico que almacena algunas de las soluciones no dominadas encontradas previamente. Este archivo es usado como un conjunto de referencia contra el cual se compara cada individuo mutado. Esto es análogo al torneo que se usa en el NPGA.

PAES también utiliza un nuevo enfoque para mantener diversidad, el cual consiste de un procedimiento de agrupamiento que divide el espacio de las funciones objetivo de una manera recursiva. Cada solución se sitúa en una rejilla de localización, con base en los valores de las funciones objetivo. Se mantiene un mapa de dicha rejilla y un conteo del número de soluciones que residen en cada retícula de la misma. Dado que el procedimiento es adaptativo, no se requieren parámetros extras en el algoritmo. Más aún, el procedimiento tiene una baja complejidad en comparación con los métodos basados en nichos [38].

3.3.4. Non-dominated Sorting Genetic Algorithm-II (NSGA-II)

En el año 2000, Deb, Agrawal, Pratap y Meyarivan, presentaron una segunda versión del algoritmo NSGA [15]. El NSGA-II heredó la estructura principal de su antecesor, pero cuenta con mejoras muy significativas que lo hacen más eficiente (computacionalmente hablando) y las cuales mejoran sustancialmente su desempeño.

Algoritmo 7: Pareto Archived Evolution Strategy (PAES)

Input: Un número máximo de generaciones max_{gen}
Output: Una población evolucionada

```
1  begin
2      Inicializar población con un solo padre p, evaluar su aptitud y agregarlo a un
       archivo externo;
3      for i = 1 to max_gen do
4          Mutar a p para producir un individuo hijo c y evaluar su aptitud;
5          case (p ⪯ c)
6              Descartar c;
7          case (c ⪯ p)
8              Reemplazar a p por c, y agregar a c al archivo;
9          case (c sea dominada por algún miembro del archivo)
10             Descartar c;
11         else
12             Aplicar la prueba para (p, c y el archivo) para determinar cuál llegará a
               ser la nueva solución actual (p) y si se incluirá a c en el archivo;
13     end
14 end
```

Las características que enmarca al algoritmo NSGA-II son:

- El ordenamiento no dominado mediante una técnica de comparación que utiliza una subpoblación auxiliar, que le permite disminuir la complejidad de ordenación de $O(MN^3)$ a $O(MN^2)$, donde M es el número de funciones objetivo y N el tamaño de la población P.
- El uso de una técnica de agrupación (*crowding*), que no requiere especificar parámetros adicionales para la preservación de diversidad en la población, eliminando la dependencia del parámetro σ_{share} (el radio de nicho) utilizado por el NSGA original.
- La asignación de valores de aptitud con base en los niveles o jerarquías de no dominancia (*rank*), aunque se considera en el procedimiento de asignación a los valores de distancia de crowding utilizados para evaluar la diversidad de las soluciones.

A continuación, describiremos las diferentes fases que componen al NSGA-II, para posteriormente describir la mecánica completa del algoritmo.

3.3.4.1. Enfoque del ordenamiento rápido no dominado

Para entender claramente este enfoque, primero describiremos un procedimiento sencillo y lento para ordenar una población, en sus distintos niveles de no dominación. Después, describiremos el enfoque rápido propuesto para el NSGA-II.

En un enfoque sencillo, para identificar las soluciones del primer frente no dominado en una población de tamaño N, se requiere comparar cada solución con el resto de la población para determinar si es dominada. Esto requiere $O(MN)$ comparaciones para cada solución, donde M es el número de funciones objetivo. Cuando este procedimiento continúa para encontrar todos los miembros del primer frente no dominado, la complejidad total es de $O(MN^2)$. Para encontrar los individuos del segundo frente no dominado, se eliminan temporalmente de la población las soluciones del primer frente y se utiliza el mismo procedimiento que en el primer caso. De esta manera, el peor caso ocurre cuando hay N frentes, es decir, cuando existe una única solución en cada frente. En este caso, se requiere un esfuerzo computacional de $O(MN^3)$ comparaciones.

En el enfoque de ordenamiento rápido no dominado, para cada individuo p de la población P se determina el valor de dos entidades: 1) un contador de dominación n_p, que indica el número de soluciones que domina la solución p y 2) un conjunto de soluciones S_p que la solución p domina. Esto requiere de $O(MN^2)$ comparaciones. Todas las soluciones en el primer frente no dominado ($\mathcal{F}_1$), tendrán su contador de dominancia n_p igual a cero puesto que ninguna solución las domina. Por cada solución p con $n_p = 0$, se visita cada miembro q del conjunto S_p y se decrementa su contador en uno. Al hacerlo, si para algún miembro q el contador de dominación llega a ser cero, este punto se agrega a una lista Q. Todos los miembros de Q pertenecen al segundo frente no dominado ($\mathcal{F}_2$). Se continúa con el procedimiento anterior con cada miembro de Q y se forma el tercer frente no dominado ($\mathcal{F}_3$). Este procedimiento se mantiene, hasta que todos los frentes ($\mathcal{F}_i$) son identificados. El pseudo-código del enfoque se muestra en el algoritmo 8, donde p_{rank} y q_{rank} denotan el rango jerárquico de las soluciones p y q, respectivamente.

3.3.4.2. Preservación de la diversidad en la población

Durante la convergencia hacia el conjunto de óptimos de Pareto, es deseable que un algoritmo mantenga una buena dispersión en el conjunto de soluciones obtenido. El NSGA utiliza el enfoque conocido como función de compartición (*sharing function*), la cual permite mantener diversidad en la población. Sin embargo, dicha función involucra asignar un valor a un parámetro conocido como σ_{share} (radio de nicho). Este parámetro está relacionado directamente con la medición de la distancia entre los individuos de la población y no existe un procedimiento sistematizado para encontrar su valor más adecuado. En el NSGA-II, este parámetro se ha sustituido por el enfoque denominado comparación de agrupamiento (*croweded-comparison*). De esta manera, no se requiere definir algún parámetro para mantener la diversidad en la población. Antes de definir el operador de comparación de agrupamiento, se definirá el concepto de estimación de densidad (*density-estimation metric*) y el operador de comparación de agrupamiento (*crowded-comparison operator*).

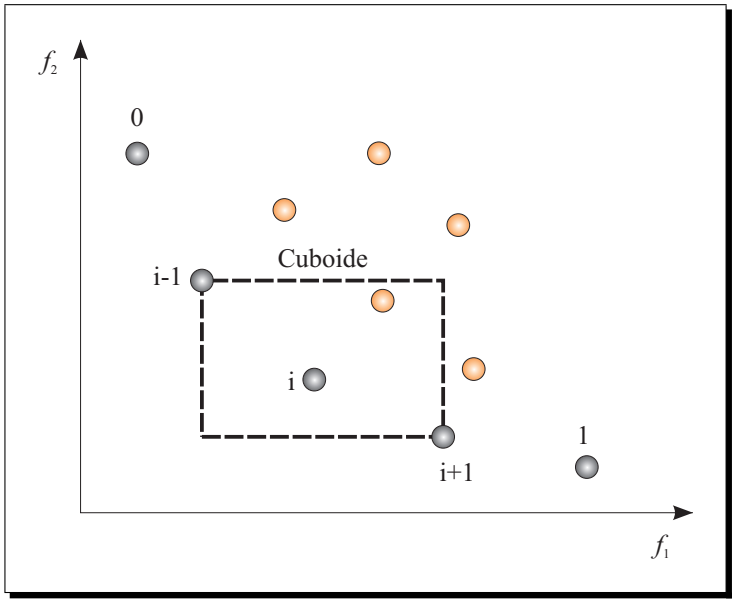

Figura 3.2: Distancia de agrupamiento

1) Estimación de densidad (*density-estimation metric*).

   Para obtener una estimación de la densidad de las soluciones circundantes en una solución en particular, se calcula la distancia promedio de dos puntos vecinos, uno en cada lado del punto de interés, utilizando el valor de sus funciones objetivo. Esta cantidad i_{dist} sirve como un estimador del perímetro del cuboide formado por los vecinos más cercanos como los vértices; a esto se le conoce como distancia de agrupamiento (*crowding distance*) y se ilustra gráficamente en la figura 3.2.

   El valor de la distancia de agrupamiento del *i*-ésimo individuo en su frente, es la longitud promedio de sus lados. El procedimiento para calcular la distancia de agrupamiento, requiere ordenar descendentemente la población de acuerdo con el valor de cada función objetivo. Después, por cada función objetivo se asigna un valor de distancia infinito a los individuos que están en los extremos (individuos con el menor y mayor valor). Al resto de las soluciones, se les asigna un valor de distancia normalizado, que es igual al valor absoluto de la diferencia de lo valores de la funciones, correspondientes a los puntos adyacentes. El valor de la distancia de agrupamiento, se calcula como la suma de los valores de todas las distancias de los elementos correspondientes a cada objetivo. Se normaliza cada función objetivo antes de calcular la distancia de agrupamiento. El algoritmo 9, muestra el procedimiento para calcular la distancia de agrupamiento de todas las soluciones en un conjunto no dominado (I). En el algoritmo 9, $I[i].m$ es el valor de la m-ésima función objetivo correspondiente al i-ésimo individuo del conjunto ordenado I. Los parámetros $f_m^{\text{mín}}$ y $f_m^{\text{máx}}$ son el máximo y mínimo valor de la m-ésima función, respectivamente. La complejidad computacional de este procedimiento está relacionada directamente con el proceso de ordenamiento, ya que se requieren M ordenaciones independientes de a lo más

N individuos. En el peor de los casos, cuando toda la población está en el frente I, la complejidad que se tiene es $O(MN\log N)$. Cuando los miembros del conjunto I tienen asignado un valor de la distancia de agrupamiento, se pueden comparar dos integrantes por su separación o proximidad con otros individuos. Un valor de distancia pequeño, indica que tiene más vecinos y, por lo tanto, es menos deseable.

2) Operador de comparación de agrupamiento (*crowded-comparison operator*).

   El operador de comparación de agrupamiento $\prec_n$, guía el proceso de selección a varias fases del algoritmo hacia un frente de Pareto óptimo y disperso. Asumiendo que cada individuo i en la población tiene dos atributos:

   i. Jerarquía de no dominación i_{rank}.

   ii. Distancia de agrupamiento $i_{distance}$.

   se define un orden parcial $\prec_n$ como:

$$i \prec_n j \quad \textbf{if } (i_{rank} < j_{rank}) \\ \textbf{or } ((i_{rank} = j_{rank}) \textbf{ and } (i_{distance} > j_{distance}))$$

   Lo anterior significa, que entre dos soluciones con distintos rangos de no dominación, se prefiere la solución con el menor rango. En otro caso, si dos soluciones pertenecen al mismo frente, entonces se prefiere la solución que está ubicada en una región de menor densidad.

   Con estas tres innovaciones (*fast non-dominated sorting*, *fast crowded distance estimation* y *crowded comparison operator*), ahora describiremos el algoritmo completo del NSGA-II.

3.3.4.3. Ciclo completo del NSGA-II

Inicialmente, se crea una población aleatoria P_0 de tamaño N. La población es ordenada con base en el principio de no dominación. A cada solución se le asigna un valor de jerarquía igual al nivel de su dominación (el nivel 1 es el mejor, el nivel 2 es el siguiente mejor y así sucesivamente). Al principio del algoritmo, se aplica la selección de torneo binario de forma usual, así como los operadores de cruza y mutación, para crear una población de descendientes Q_0 de tamaño N. El concepto de elitismo se aplica al comparar la población actual con las mejores soluciones previamente encontradas (selección $(\mu+\lambda)$). El procedimiento difiere después de producir la generación inicial tal y como se describe a continuación con la t-ésima generación.

En primera instancia, se obtiene una población formada por $R_t = P_t \cup Q_t$ de tamaño $2N$. Después, la población R_t se ordena de acuerdo a los principios de no dominación. Dado que todos los miembros de la población previa y actual están incluidos en R_t, el elitismo está asegurado. Las soluciones no dominadas pertenecientes al conjunto $\mathcal{F}_1$ son las mejores

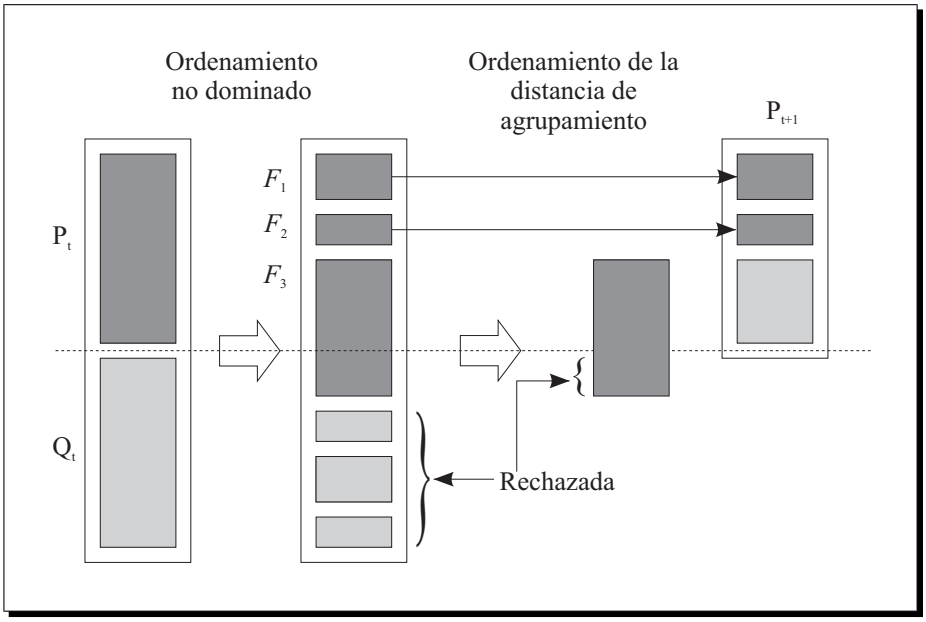

Figura 3.3: Procedimiento del NSGA-II

soluciones de la población R_t y deben ser enfatizadas más que alguna otra solución en la población combinada. Si el tamaño de $\mathcal{F}_1$ es más pequeño que N, se eligen todos los miembros del conjunto $\mathcal{F}_1$ para ser parte de la población P_{t+1}. Los individuos restantantes de la población P_{t+1} son elegidos de los subsecuentes frentes no dominados en el orden de su rango. Así que las soluciones del conjunto $\mathcal{F}_2$ son elegidas, seguidas por las soluciones del conjunto $\mathcal{F}_3$ y así sucesivamente. Este procedimiento continúa hasta que no pueden acomodarse más conjuntos de soluciones. Evidentemente, la suma de las cardinalidades de los conjuntos $\mathcal{F}_1$ a $\mathcal{F}_l$ es mayor que N. Para solucionar esta situación, se ordena en forma descendente el último conjunto F_l usando el operador de comparación de agrupamiento ($\prec_n$) para seleccionar las mejores soluciones y llenar exactamente el espacio de la nueva población de tamaño N. El NSGA-II se describe gráficamente en la figura 3.3. A la nueva población P_{t+1} se le aplican los operadores de selección, cruza y mutación para formar una nueva población Q_{t+1} de tamaño N. Es importante decir que en la técnica de selección de torneo binario, se aplica el operador de comparación de agrupamiento $(\prec_n)$. Este operador requiere que se conozca la jerarquía y la distancia de agrupamiento de cada solución de la población. Estos últimos valores se calculan mientras se está formando la población P_{t+1} tal y como se muestra en el algoritmo 10.

Algoritmo 8: Enfoque del ordenamiento rápido no dominado

Input: Una población P

Output: Una asignación jerárquica a los miembro de la población, con base en sus niveles de dominancia, clasificados en los distintos frentes $\mathcal{F}_i$

```
1  begin
2      forall (p ∈ P);                    /* para cada individuo p de la población P */
3      do
4          S_p = ϕ;
5          n_p = 0;
6          forall (q ∈ P) ;               /* para cada individuo p de la población P */
7          do
8              if (p ≺ q);                                  /* si p domina a q */
9              then
10                 S_p = S_p ∪ {q}; /* S_p el conjunto de soluciones que son donminadas
                   por p */
11             else if (q ≺ p);                             /* si q domina a p */
12             then
13                 n_p = n_p + 1;          /* contador de soluciones que dominan a p */
14         end
15         if (n_p = 0);                              /* si nadie domina a p */
16         then
17             p_rank = 1;                /* se asigna primer grado de jerarquía a p */
18             F_1 = F_1 ∪ {p} ;                  /* F_1 primer frente encontrado */
19         end
20     end
21     i = 1;
22     while (F_i ≠ ϕ);                    /* se obtienen los F_i frentes restantes */
23     do
24         Q = ϕ;
25         forall (p ∈ F_i);                /* para cada individuo p del frente F_i */
26         do
27             forall (q ∈ S_p);           /* para cada individuo q del conjunto S_p */
28             do
29                 n_q = n_q − 1;
30                 if (n_q = 0) then
31                     q_rank = i + 1;  /* asigna el siguiente nivel jerárquico a q */
32                     Q = Q ∪ {q}
33             end
34         end
35         i = i + 1;
36         F_i = Q;
37     end
38 end
```

Algoritmo 9: Asignación de la distancia de agrupamiento

Input: Un conjunto no dominado I
Output: Una asignación de distancia de agrupamiento a cada miembro de I

begin
  $l = |I|$;
  forall $(i \in I)$ **do**
    $I[i]_{distance} = 0$;
  end
  forall *objetivo m* **do**
    $I = sort(I, m)$;
    $I[i]_{distance} = I[l] = \infty$;
    for $i = 2$ **to** $l - 1$ **do**
      $$I[i]_{distance} = I[i]_{distance} + \frac{I[i+1].m - I[i-1].m}{f_m^{\text{máx}} - f_m^{\text{mín}}};$$
    end
  end
end

Algoritmo 10: Non-dominated Sorting Genetic Algorithm II (NSGA-II)

Input: Un número máximo de generaciones max_{gen}
Output: Una población evolucionada

begin
  $t = 0$;
  Inicializar población P_t;
  Evaluar población de acuerdo a las funciones objetivo;
  while $(t < max_{gen})$ **do**
    Aplicar cruza y mutación a P_t, para generar Q_t;
    Evaluar población Q_t de acuerdo a las funciones objetivo;
    $R_t = P_t \cup Q_t$;
    Asignar jerarquía con base en la dominancia de Pareto a R_t;
    Asignar distancia de agrupamiento a R_t;
    Seleccionar los N individuos de R_t, de acuerdo al operador de comparación de agrupamiento $\prec_n$ para obtener P_{t+1};
    $t = t + 1$;
  end
end

CAPÍTULO 4

PROPUESTA MONO-OBJETIVO

Como trabajo previo al principal problema que se aborda en esta tesis (optimización multiobjetivo), presentamos un algoritmo híbrido al que denominamos *Nonlinear Simplex Search Differential Evolution* (NSSDE) para optimización global. Este enfoque utiliza el algoritmo evolutivo de evolución diferencial [57], acoplado al método de Nelder-Mead [53], que actúa como buscador local. Nuestra idea es diseñar una estrategia de búsqueda local, que pueda ser utilizada de manera similar en el problema de optimización multiobjetivo.

En el resto del capítulo, describiremos la heurística de optimización global adoptada, y presentaremos algunos trabajos referentes a algoritmos híbridos para optimización global, que utilizan el método de Nelder-Mead en su hibridización. También presentaremos una sección donde se exponen los resultados de nuestra propuesta y las conclusiones sobre los resultados obtenidos del algoritmo híbrido.

4.1. Evolución diferencial

La *evolución diferencial* (ED) es una rama de la computación evolutiva desarrollada por Rainer Storn y Kenneth Price [71] a mediados de los 1990s. Surge como consecuencia de los intentos de Price por resolver el polinomio de Chebyshev y es utilizada para resolver problemas de optimización multidimensionales en espacios continuos. La idea que Price concibió, es utilizar la diferencia entre vectores para perturbar a otro vector de la población. Con esta idea seminal, una intensa discusión surgió entre Price y Storn, iniciada en interminables meditaciones y simulaciones por computadora, lo que produjo un gran número de mejoras sustanciales, las cuales hicieron de la ED la herramienta versátil y robusta de optimización que es hoy en día.

La “comunidad de ED” ha estado creciendo desde sus orígenes (1994–1995) y cada día más investigadores trabajan con esta técnica. El deseo de Price y Storn, es que sea

fomentado el desarrollo de esta heurística por científicos en todo el mundo y pueda ser mejorada para ayudar a más usuarios en su trabajo. Es por esto, que la ED no ha sido patentada de ninguna forma.

4.1.1. Optimización global con ED

La ED es un método de búsqueda que utiliza N vectores:

$$x_{i,g}; i = \{1,2,\ldots,N\}$$

como la población de cada generación g. El valor de N representa la cardinalidad de la población y es constante durante todo el proceso de optimización. La población inicial se elige de manera aleatoria si no se conoce nada acerca del problema. Como regla, se asume una distribución uniforme para las decisiones aleatorias a tomar. La idea principal de la ED es un nuevo esquema para crear vectores. De esta manera, los nuevos vectores son generados cuando se suma la diferencia de pesos de dos de ellos a un tercer vector, todos ellos miembros de la población. Si la aptitud del vector resultante es mejor que el miembro de la población elegido, entonces el nuevo vector reemplaza al vector con el cual fue comparado.

De acuerdo a los estudios realizados por Price y Storn existen diferentes variantes en ED. Las dos variantes más prometedoras son ED1 y ED2 [71], las cuales describimos a continuación.

4.1.1.1. Esquema ED1

Para cada vector $\vec{x}_{i,G}; i = \{1,2\ldots,N\}$, un vector de prueba $\vec{v}$ se genera de acuerdo a:

$$\vec{v} = \vec{x}_{r_1,g} + F \cdot (\vec{x}_{r_2,g} - \vec{x}_{r_3,g}) \tag{4.1}$$

con $r_1, r_2, r_3 \in [1,N]$ enteros y $F > 0$.

Los enteros r_1, r_2 y r_3 son elegidos aleatoriamente en el intervalo $[1,N]$ y son diferentes entre sí. La constante F, es un factor real que controla la amplificación en la variación diferencial $(\vec{x}_{r_2,g} - \vec{x}_{r_3,g})$. Este esquema se ilustra gráficamente en la figura 4.1.

4.1.1.2. Esquema ED2

Este esquema trabaja de igual forma que el ED1, pero genera el vector $\vec{v}$ de acuerdo a:

$$\vec{v} = \vec{x}_{i,g} + \lambda \cdot (\vec{x}_{mejor,g} - \vec{x}_{i,g}) + F \cdot (\vec{x}_{r_2,g} - \vec{x}_{r_3,g}) \tag{4.2}$$

introduciendo una variable de control adicional λ. La idea es incorporar al esquema el mejor vector de la población $x_{mejor,g}$. Esta característica puede ser muy útil para las

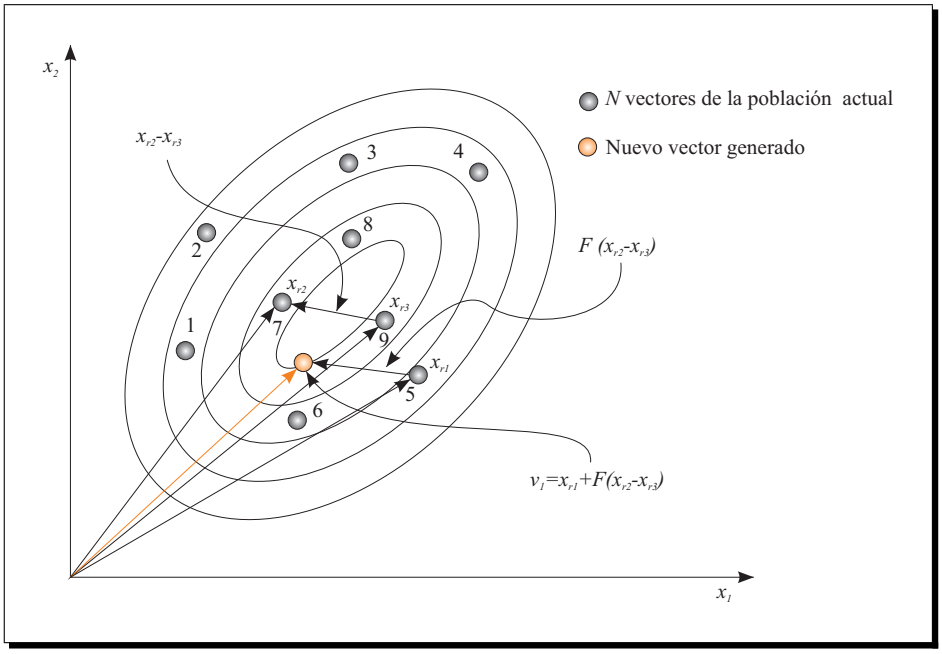

Figura 4.1: Esquema ED1

funciones no críticas.

Para incrementar la diversidad de los resultados, se usa un vector:

$$\vec{u} = (u_1, u_2, \ldots, u_D)^T$$

con:

$$u_j = \begin{cases} v_j & \text{, para } j = \langle n \rangle_D, \langle n+1 \rangle_D, \ldots, \langle n+L-1 \rangle_D \\ (x_{i,g})_j & \text{, de lo contrario.} \end{cases}$$

donde D indica el espacio euclidiano de las variables de la función y los paréntesis $\langle\rangle_D$ denotan la función módulo con módulo D.

En los dos esquemas ($ED1$ y $ED2$) puede surgir el problema, de que al generar el vector hijo $\vec{u}$, los componentes del vector excedan el límite de las variables. Si este es el caso, entonces se debe adoptar una estrategia para restablecer el parámetro en un valor permitido.

El método más simple, es hacer que el parámetro u_j tome el valor del límite que excede. Aunque en la práctica este método ha trabajado bien, podría ser evitado, dado que esto tiene el potencial de decrementar la diversidad de la población. Otra alternativa, es que si el parámetro está fuera de los límites, puede restaurarse reinicializándose (es decir, generando un nuevo valor aleatorio). Mientras que esta estrategia mantiene la diversidad de los parámetros, es quizá, una forma un tanto extrema de preservar diversidad.

Una buena idea es la que presenta Price en [56] y es la tomada en este trabajo. Consiste en restablecer los parámetros que violan las restricciones a un punto medio entre su premutación/cruza y el valor del límite violado. La siguiente ecuación muestra cómo usar este enfoque para restablecer el j-ésimo parámetro del i-ésimo vector hijo $u_{j,i,g+1}$:

$$u_{j,i,g+1} = \begin{cases} \left(x_{j,i,g} + x_j^{(lo)}\right)/2 & \text{, si } u_{j,i,g+1} < x_j^{lo} \\ \left(x_{j,i,g} + x_j^{(hi)}\right)/2 & \text{, si } u_{j,i,g+1} < x_j^{hi} \\ u_{j,i,g+1} & \text{, en otro caso.} \end{cases}$$

donde *lo* y *hi* indica el límite inferior y superior permitido, respectivamente.

4.1.2. Estrategias de ED

Además de los esquemas de ED, existen diferentes estrategias que pueden adoptarse en el algoritmo y son utilizadas dependiendo del tipo de problema al que se aplique. Las estrategias recaen en el vector a perturbar, en el número de diferentes vectores considerados para la perturbación, y finalmente en el tipo de cruza a utilizar. Las diez estrategias propuestas por Price y Storn [72, 57] son las siguientes:

1. ED/*mejor*/1/exp
2. ED/*aleatorio*/1/exp
3. ED/*aleatorio-al-mejor*/1/exp
4. ED/*mejor*/2/exp
5. ED/*aleatorio*/2/exp
6. ED/*mejor*/1/bin
7. ED/*aleatorio*/1/bin
8. ED/*aleatorio-al-mejor*/1/bin1
9. ED/*mejor*/2/bin
10. ED/*aleatorio*/2/bin

De esta forma, la convención utilizada anteriormente es: *ED/x/y/z*.

- *ED*

  Referente a la evolución diferencial.

- *x*

  Representa el vector a perturbar; puede ser el mejor vector de la generación anterior (*mejor*) o cualquier vector elegido aleatoriamente (*aleatorio*).

- *y*

  Es el número de vectores considerados para la perturbación del vector *x*. Para la perturbación de un simple vector, se eligen aleatoriamente 3 vectores diferentes; la diferencia de dos de ellos se suma al tercero. Para perturbar dos vectores diferentes, se eligen 5 vectores distintos; la diferencia de pesos para cada par de cualquiera de los cuatro primeros vectores se suma al quinto.

- *z*

  Es el tipo de cruza que puede ser utilizada (exp: *exponencial*; bin: *binomial*). La cruza en la ED se realiza en las variables del vector dentro de un ciclo limitado por el valor de *CR*. En la cruza exponencial, la primera vez en la que un número aleatorio entre $(0,1)$ supera el valor de *CR* se suspende este ciclo y las variables que queden por alterar quedan intactas. En la cruza binomial, la cruza se realiza en cada una de las variables siempre y cuando al elegir un número aleatorio entre $(0,1)$ éste sea menor que el valor de *CR*.

Las dos variantes de este algoritmo evolutivo, aunadas a las estrategias propuestas por Storn y Price, hacen de la evolución diferencial, una técnica heurística competitiva en problemas de optimización global. Tanto la estrategia a adoptar como los parámetros *CR* y F, dependen directamente del problema que se enfrente y se determina independientemente mediante *ensayo-y-error*. En la figura 4.2 se ilustra claramente el procedimiento para generar un nuevo vector y se muestra cómo determinar si pasa o no a la siguiente generación. Por último, en el algoritmo 11 presentamos el pseudo-código del algoritmo de evolución diferencial de acuerdo a la estrategia *ED1/aleatorio/1/bin*.

4.2. Trabajos relacionados

En la actualidad existe una gran variedad de algoritmos híbridos para resolver problemas de optimización global. Estos algoritmos han sido desarrollados combinando métodos de distintas naturalezas. Los que conciernen a este trabajo, son los trabajos que utilizan el algoritmo de Nelder-Mead en su hibridización.

A continuación mencionaremos brevemente algunos trabajos recientes, que han adoptado el método de Nelder-Mead como buscador local, para resolver problemas optimización global.

- En 2005, Fang Wang y Yuhui Qiu proponen el algoritmo híbrido *Nonlinear Simplex Search PSO* (NSSPSO) [82], el cual está basado en el PSO canónico (una variante del PSO original) propuesto por Carlisle y Dozier [7]. El algoritmo utiliza el método de Nelder-Mead como buscador local. La idea general del NSSPSO, consiste en aislar una partícula $\vec{x}_i$ del cúmulo y aplicar el método de Nelder-Mead sobre ella. Si durante la ejecución de la búsqueda local, la partícula converge en un radio de tolerancia

Algoritmo 11: Evolución diferencial estrategia: *ED1/aleatorio/1/bin*

Input: Un número máximo de generaciones max_{gen}
Output: Una población evolucionada $x_{i,g}$

begin
  $g = 0$;
  Iniciar una población aleatoria $x_{i,g}$ de tamaño N;
  Evaluar la aptitud de los individuos de la población;
  while $(g < max_{gen})$ **do**
    for $i = 1$ ***to*** N **do**
      Elegir $r_1 \neq r_2 \neq r_3$ aleatoriamente;
      $j_{rand} = rand\_integer[1, D]$;
      for $j = 1$ **to** D **do**
        if $(rand_j[0,1) < CR)$ ***or*** $(j = j_{rand})$ **then**
          $u_{i,j,g+1} = x_{r_3,j,G} + F \cdot (x_{r_1,j,g} - x_{r_2,j,g})$;
        else
          $u_{i,j,g+1} = x_{j,g}$;
        end
      end
      if $(f(u_{i,g+1}) \leq f(x_{i,g}))$ **then**
        $x_{i,g+1} = u_{i,g+1}$;
      else
        $x_{i,g+1} = u_{i,g}$;
      end
    end
    $g = g + 1$;
  end
end

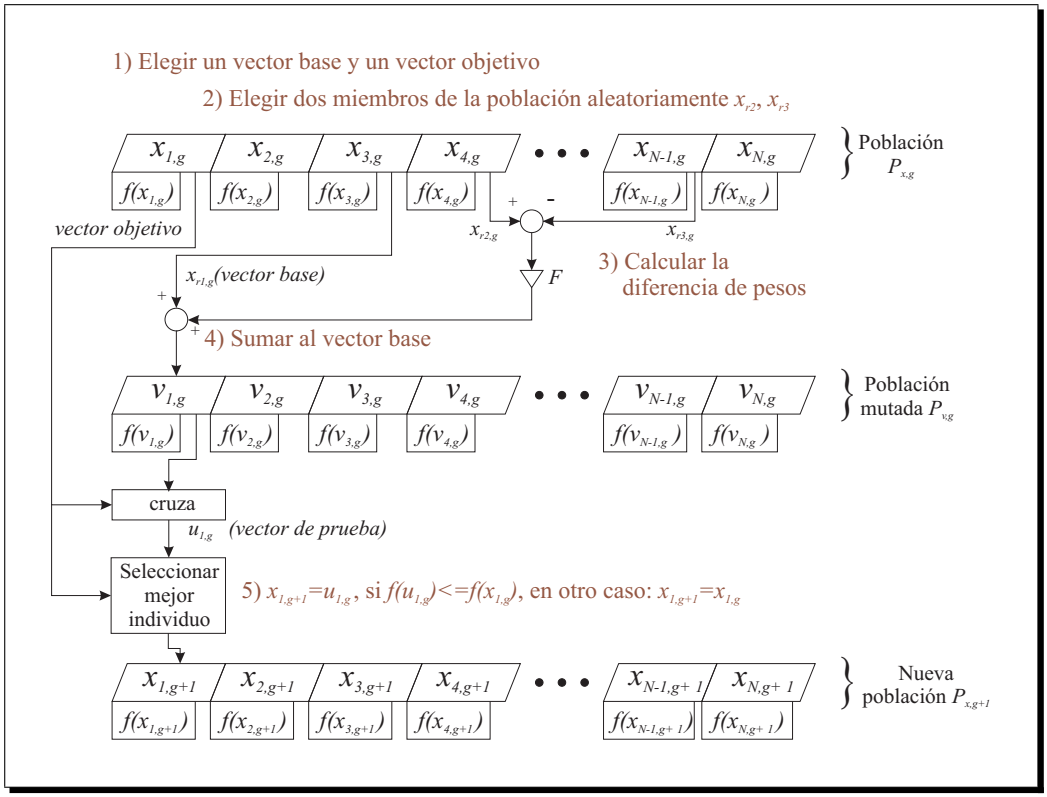

Figura 4.2: Generación de vectores utilizando evolución diferencial con base a la estrategia (*ED1/aleatorio/1/bin*)

(de magnitud *ErrorGoal*) establecido, la iteración del PSO es considerada como satisfactoria; de lo contrario, la partícula puede ser devuelta al cúmulo e iniciar con la siguiente iteración del PSO. El simplex inicial que utiliza Nelder-Mead, se construye a partir de la partícula aislada $\vec{x}_i$. El resto de los vértices son generados aleatoriamente de acuerdo a una distribución normal, con media x_i y desviación estándar igual a *DRadius*, donde:

$$DRadius = ErrorGoal + \delta$$

$$\delta = \begin{cases} 100 * ErrorGoal & , \text{si } ErrorGoal \leq 10^{-4} \\ 0.001 * ErrorGoal & , \text{de lo contrario.} \end{cases}$$

y $ErrorGoal$ es una constante definida de acuerdo al problema de optimización.

- Otra propuesta que utiliza a Nelder-Mead como buscador local, es el algoritmo híbrido que presenta Koduru, Das y Welch en 2006 [40]. Esta propuesta también está basada en el algoritmo del PSO y presenta dos esquemas de hibridización. El primero denominado "*tandem*", consiste en dividir la población en dos subconjuntos (para cada iteración). Mientras que el primer conjunto está sujeto al algoritmo principal estocástico (PSO), una o más iteraciones del algoritmo de Nelder-Mead son aplicadas al otro subconjunto. Al final, ambos subconjuntos son mezclados y se continúa con la

siguiente iteración. El segundo esquema es denominado "*cascade*". En esta propuesta, la idea es aplicar el proceso de optimización estocástico a todos los individuos de la población y mejorar más las soluciones obtenidas utilizando Nelder-Mead. Para los dos esquemas, la definición del simplex depende de la partícula elegida para optimizar, ya que para definir el resto de los vértices del simplex se aplica una técnica de *clustering* (*K-means*) [46], con la que se obtienen las partículas de la población más cercanas a ella. De acuerdo a los experimentos que realizan los autores, concluyen que el esquema *cascade* da mejores resultados.

- Recientemente en 2007, Luo y Yu proponen el algoritmo *low dimensional simplex evolution* (LDSE) [45], el cual es un algoritmo híbrido basado en la evolución diferencial y el algoritmo de Nelder-Mead. En la LDSE, la cruza/mutación que se realiza en la ED para crear a un nuevo hijo, se sustituye por los movimientos que realiza el algoritmo de Nelder-Mead. La idea general consiste de dos sencillos pasos: 1) inicializar una población P de tamaño N y evaluar la aptitud; 2) para cada individuo $\vec{x}_i$ de la población P, se seleccionan aleatoriamente $m+1$ individuos de P, los cuales representan los vértices del simplex. El algoritmo de Nelder-Mead se ejecuta sobre el m-simplex definido, de tal forma que si algún movimiento del algoritmo de Nelder-Mead genera un individuo con mejor aptitud que el individuo $\vec{x}_i$, se sustituye. En caso contrario, si la aptitud del individuo $\vec{x}_i$ es peor o igual que la aptitud promedio de la población P y además es peor que el mejor vértice ($\vec{x}_b$) del m-simplex, al vector $\vec{x}_i$ se le asigna un punto intermedio entre $\vec{x}_b$ y $\vec{x}_i$; si por el contrario, la aptitud de $\vec{x}_i$ es mejor que $\vec{x}_b$, entonces se asigna a $\vec{x}_i$ un vector intermedio entre $\vec{x}_i$ y el peor vértice ($\vec{x}_w$) del m-simplex. Recibe el nombre de LDSE debido a que cuando se elige el valor de m, tal que $m < n$ (donde, n es la dimensionalidad de la función), la dimensión del simplex se reduce y en estas condiciones, el simplex evoluciona en una dimensión menor que la de la función. Aunque los autores del algoritmo analizan el caso cuando $m = n$ (*full dimensional simplex evolution* (FDSE)), los resultados que obtienen al comparar las dos propuestas, hacen concluir que la LDSE resulta más eficiente.

4.3. Nuestra propuesta

En términos de evolución, la búsqueda local es una consecuencia del aprendizaje adquirido a lo largo de la vida de un individuo. La aptitud del individuo cambiará después de aplicar el proceso de aprendizaje. Sin embargo, los códigos genéticos del individuo (*parámetros*) pueden o no ser alterados dependiendo la manera en que interactúe el aprendizaje en la evolución. A esto se le conoce como evolución Lamarckiana y efecto Baldwin [29, 2], respectivamente.

Las técnicas de búsqueda local han sido aplicadas en distintas heurísticas de optimización. La manera más simple para que dichas heurísticas puedan ser beneficiadas por la búsqueda local, es utilizando los Métodos Multiarranque (*Multistart Methods*) [47]. En términos generales, estos métodos consisten en alternar una fase de generación de

soluciones con otra de mejora de las mismas. El proceso se repite hasta que se cumpla un criterio de parada. Se caracteriza principalmente porque se realizan múltiples búsquedas locales en distintas soluciones, con el fin de evitar caer en zonas donde existan soluciones localmente óptimas.

En nuestra propuesta, a la que denominamos *Nonlinear Simplex Search Differential Evolution* (NSSDE), el mecanismo de aprendizaje adoptado se basa en la teoría de la evolución Lamarckiana. A diferencia de los métodos multiarranque, nosotros realizamos la búsqueda local únicamente a una solución de la población. La idea es suministrar un procedimiento que combine adecuadamente el poder de exploración del método de construcción de soluciones (algoritmo evolutivo), con el poder de explotación del método de mejora (algoritmo de Nelder-Mead).

De esta manera, las dos fases que componen el algoritmo híbrido propuesto son: 1) Fase de exploración y 2) Fase de explotación. A continuación describiremos estas fases para posteriormente describir la mecánica completa del algoritmo.

4.3.1. Fase de exploración

Este mecanismo tiene como objetivo generar diversas soluciones en distintas zonas del espacio de búsqueda. El método que utilizamos para esta fase, es el algoritmo de evolución diferencial descrito anteriormente.

El objetivo principal de adoptar un algoritmo evolutivo, es mantener una amplia diversidad de soluciones durante el proceso de optimización. De esta manera, la probabilidad de caer en zonas de óptimos locales disminuye. Además, es posible ahorrar evaluaciones de la función en el método de mejora, debido a que a diferencia de los métodos multiarranque, nosotros sólo aplicaremos una búsqueda local, en una única solución.

4.3.2. Fase de explotación

Como hemos mencionado, en esta fase nosotros adoptamos el método de Nelder-Mead. La idea es elegir un individuo de la población para mejorarlo. Dicho individuo cambiará su estructura genética y reemplazará al peor miembro de la población actual para, de esta manera, dar paso a la siguiente generación.

El aplicar una búsqueda local en un algoritmo evolutivo, trae consigo dos interrogantes: 1) ¿Cuál es la frecuencia con la que se debe aplicar el método de mejora en el algoritmo? y 2) ¿Cuál debe ser el criterio de parada del mismo?.

En las siguientes secciones presentaremos los criterios tomados para estas dos inquietudes.

4.3.2.1. Criterio de frecuencia para aplicar la búsqueda local

Aplicar el buscador local generación tras generación del algoritmo evolutivo, implicaría un número excesivo de evaluaciones de la función objetivo. El criterio que hemos adoptado en este trabajo, se basa en la probabilidad de recombinación del individuo evolucionado x^* (por la búsqueda local), con cualquier otro individuo de la población P de tamaño N.

La probabilidad de que el individuo x^*, pueda ser seleccionado para la cruza sexual con otro miembro de la población, está dada por $\frac{1}{N}$. Si aplicáramos el método de mejora cada N iteraciones, la probabilidad de que el individuo x^* herede los rasgos genéticos puros a cada miembro de la población debería ser igual a 1. Sin embargo, dado el individuo x^*, éste cambia su estructura genética por efecto de los operadores evolutivos de la evolución diferencial. Nosotros proponemos aplicar el método de mejora con una frecuencia de $\frac{N}{2}$ generaciones, con el fin de intensificar y reproducir el aprendizaje del individuo mejorado en la población.

4.3.2.2. Criterio de parada de la búsqueda local

En el libro de Rao [59], se asume que el algoritmo de Nelder-Mead ha convergido siempre y cuando, la desviación estándar de la función en los $n+1$ vértices del simplex (n la dimensión de la función), sea menor que un valor suficientemente pequeño ε, esto es:

$$Q = \left\{ \sum_{i=0}^{n} \frac{[f(\Delta_i) - f(\vec{x}_c)]^2}{n+1} \right\} \leq \varepsilon \tag{4.3}$$

donde $\vec{x}_c$ es el centroide del simplex y Δ_i los vértices del mismo.

Treinta años después de la publicación del algoritmo de Nelder-Mead, Lagarias et al. presentan un trabajo [43] detallado sobre las propiedades y convergencia del método. Antes de mencionar algunas propiedades que son tomadas en este trabajo, veamos la siguiente definición:

DEFINICIÓN 11 (SIMPLEX NO-DEGENERADO)
Se dice que un simplex es *no-degenerado*, si y sólo si, los vectores del simplex forman un conjunto linealmente independiente. En otro caso, se dice que el simplex es *degenerado* y en estas circunstancias, se tendrá también un simplex pero de dimensión menor.

Lagarias demuestra que todos los movimientos que puede realizar el algoritmo de Nelder-Mead siempre generan un nuevo simplex *no-degenerado*. Además, muestra que el método converge a un punto mejor que el peor vértice del simplex en al menos $n+1$ iteraciones del algoritmo.

Otra propiedad inmediata que surge a partir de la definición del método, es referente al número de evaluaciones que se realizan de la función objetivo. En una iteración de

Nelder-Mead, se requiere una evaluación de la función siempre que el algoritmo termine en el movimiento de reflexión; dos evaluaciones son requeridas si se acepta el movimiento de expansión o contracción; el peor de los casos ocurre cuando se acepta reducir el simplex ya que en ese caso se requieren $n+2$ evaluaciones de la función objetivo. Aunque los estudios experimentales realizados por Torczon [74] revelan que en esencia los movimiento de reducción nunca ocurren, Lagarias demuestra que únicamente no se llevan a cabo, si el método de Nelder-Mead se aplica a funciones estrictamente convexas.

El utilizar el criterio de paro de la ecuación 4.3 en la búsqueda local, implica realizar una evaluación más de la función objetivo en cada iteración del algoritmo (debido a que el centroide x_c cambia en cada iteración). El objetivo principal de nuestro trabajo es reducir el número de evaluaciones de la función. Por esta razón, y con base en lo descrito anteriormente, nosotros proponemos utilizar una condición de paro que no requiere ninguna evaluación adicional y se menciona a continuación.

Este criterio es tomado directamente de las aportaciones que expone Lagarias en su trabajo. Consiste en determinar si Nelder-Mead ha obtenido un vector mejor que el peor vértice del simplex en al menos $n+1$ iteraciones; si no es así, se concluye que el algoritmo ha convergido y se detiene la búsqueda. Pero el converger a mejores puntos no indica que el funcionamiento de Nelder-Mead sea el más adecuado, esto debido a que la convergencia puede ser demasiado lenta y demandar un número considerable de evaluaciones de la función. Por esta razón, nosotros establecemos una condición más de paro, la cual hace referencia a una tolerancia de convergencia en un determinado número de iteraciones. Es decir, si Nelder-Mead no converge más allá de una tolerancia ε en $2(n+1)$ iteraciones (propuesta con base en ensayos experimentales), entonces deducimos que el simplex ha sido modificado a tal grado que su uso resulta ineficiente. La siguiente expresión generaliza el criterio de paro que se adopta en nuestra propuesta:

$$\begin{aligned} &\textbf{if } (\ (\textit{no se genera un vector mejor que } \vec{x}_w \textit{ en } (n+1) \textit{ iteraciones}) \textbf{ or} \\ &\quad (\textit{la convergencia en } 2(n+1) \textit{ iteraciones} \leq \varepsilon)\ ) \textbf{ then} \\ &\qquad \text{Termina búsqueda local;} \end{aligned} \tag{4.4}$$

donde $\vec{x}_w$ es el peor vértice del simplex, n es la dimensión de la función y ε es un valor de tolerancia permitido para la convergencia del algoritmo en $2(n+1)$ iteraciones; para este trabajo, $\varepsilon = 1.0 \times 10^{-3}$.

4.3.3. El algoritmo

El paso inicial es crear una población aleatoria P de tamaño N. Luego aplicamos una iteración del proceso evolutivo de ED a cada individuo de P para obtener una nueva población P^*. Después, se elige al mejor vector $\vec{x}_b$ de la población P^*. A partir de $\vec{x}_b^*$ se construye el simplex regular y se aplica el método de mejora (Nelder-Mead) hasta que el criterio de parada de la ecuación 4.4 se cumpla. El mejor vértice $\vec{x}_b^*$ encontrado por Nelder-Mead, reemplazará al peor vértice $\vec{x}_w$ de P^*. Se actualiza la población $P = P^*$ y se continúa con

la siguiente iteración si es que no se ha sobrepasado el número de iteraciones permitidas. Para aplicar nuevamente el proceso de mejora, se debe verificar si se ha evolucionado por $\frac{N}{2}$ generaciones después de la última aplicación de la búsqueda local. Esta explicación se ilustra gráficamente en la figura 4.3 y el algoritmo 12 muestra el psudo-código completo del algoritmo híbrido NSSED.

Algoritmo 12: Nonlinear Simplex Search Differential Evolution (NSSDE)

Input: Un número máximo de generaciones max_{gen}
Output: Una población evolucionada P

```
1  begin
2      g = 0;
3      Iniciar una población aleatoria P de tamaño N;
4      Evaluar la aptitud de los individuos de la población;
5      while (g < max_gen) do
6          Aplicar una iteración de ED a la población P para generar P*;
7          if ((g mod N/2) = 0) then
8              Identificar el mejor vector x_b de la población P*;
9              Construir el simplex Δ_p a partir de x_b;
10             Aplicar Nelder-Mead utilizando el simplex Δ_p para obtener x*_b;
11             Sustituir el peor vector x_w de P* por x*_b;
12         end
13         P = P*;
14         g = g + 1;
15     end
16 end
```

4.4. Resultados obtenidos

Las funciones de prueba utilizadas para evaluar nuestro algoritmo híbrido NSSDE son las presentadas por Yao et al. [83]. Consiste en un conjunto de veintitrés problemas de optimización global con alta dimensionalidad y multimodalidad (ver apéndice A).

La implementación del algoritmo híbrido para optimización global, se llevó a cabo con el fin de observar en que tanto beneficia el método de Nelder-Mead al algoritmo evolutivo. Por esta razón, las comparaciones se realizaron únicamente entre la ED y la NSSDE. Para la comparación de los algoritmos, se realizaron 50 ejecuciones para cada función de prueba. En cada ejecución, se utilizó la misma semilla generadora de números pseudo-aleatorios y los mismos parámetros.

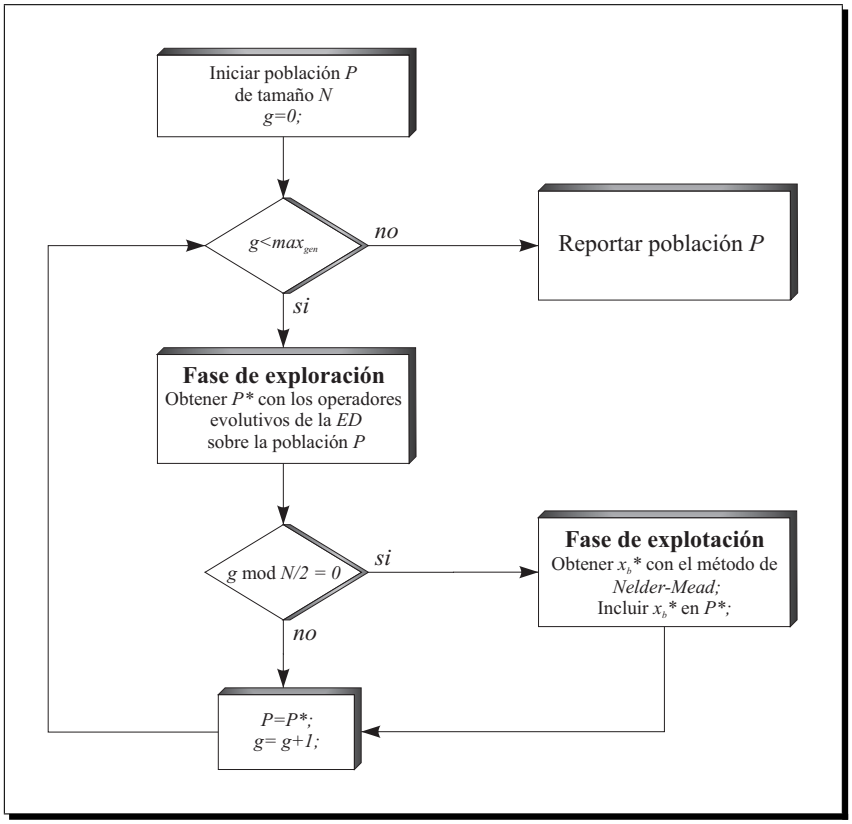

Figura 4.3: Esquema general del algoritmo híbrido NSSDE

Los resultados obtenidos en la tabla 4.1, muestran que la búsqueda local favoreció considerablemente en la mayoría de las funciones de prueba; donde "*óptimo*" es el óptimo de la función, "*Eval.*" representa el número de evaluaciones de la función, "*Des. Est.*" la desviación estándar de los mínimos encontrados y "f_{min}" el valor mínimo encontrado. La mejora que hace la NSSDE sobre la ED simple puede verse también gráficamente en las figuras 4.4 y 4.5.

4.5. Conclusiones sobre los resultados obtenidos

La estrategia utilizada en la fase de explotación fue favorable para nuestro algoritmo, en la mayoría de las funciones de prueba. Gráficamente se observa esta afirmación en las figuras 4.4 y 4.5. En ellas, solamente en las funciones 2, 8, 12 y 19, nuestro algoritmo híbrido NSSDE no supera a la ED.

La fase de explotación de nuestro algoritmo híbrido, depende directamente del punto inicial de la búsqueda. Cuando se tiene un espacio de búsqueda inmenso, la probabilidad de que el punto inicial de búsqueda esté cerca de un punto estacionario es alta. Si éste es el caso, la fase de explotación logrará rápidamente su objetivo en un número reducido

	óptimo	ED			NSSDE		
Función	0	Eval.	Des. Est.	f_{min}	Eval.	Des. Est.	f_{min}
f_1	0	20518.7	2.03E-006	9.78E-005	13185.7	2.00E-006	9.76E-005
f_2	0	25887.6	1.62E-006	9.80E-005	26154.2	1.84E-006	9.80E-005
f_3	0	47772.2	1.18E-006	9.87E-005	34238.9	1.11E-006	9.86E-005
f_4	0	80822.9	3.30E-006	9.60E-005	64748.2	2.77E-006	9.63E-005
f_5	0	192877.8	1.70E+000	9.57E-001	142799	5.58E-001	7.98E-002
f_6	0	8439.7	0.00E+000	0.00E+000	7426.3	0.00E+000	0.00E+000
f_7	0	60416.1	1.03E-003	8.71E-003	35084.9	1.05E-003	8.74E-003
f_8	-12569.5	43935.4	1.66E+001	-1.26E+004	57990.9	1.66E+001	-1.26E+004
f_9	0	275938.5	7.21E-006	8.99E-005	135341.2	2.36E-001	5.98E-002
f_{10}	0	31643.7	3.66E-006	9.55E-005	31562.8	3.40E-006	9.62E-005
f_{11}	0	26569.2	8.40E-006	9.01E-005	24314.8	1.06E-005	8.95E-005
f_{12}	0	21622.6	6.56E-006	9.25E-005	22908.8	8.89E-006	8.82E-005
f_{13}	0	23486.2	7.03E-006	9.13E-005	22761	6.43E-006	9.25E-005
f_{14}	1	1141.1	5.06E-004	9.98E-001	1121.1	5.06E-004	9.98E-001
f_{15}	0.0003075	48655.8	2.66e-008	0.00030756	43845.8	2.40e-08	0.00030756
f_{16}	-1.0316285	2317.7	2.76E-007	-1.0316	2314.6	2.69E-007	-1.0316
f_{17}	0.398	691.5	6.30E-005	0.3980	669.4	6.35E-005	0.3980
f_{18}	3	513.2	2.92E-003	3.0052	501.7	3.10E-003	3.0032
f_{19}	-3.86	335.6	7.18E-004	-3.8610	359.2	7.47E-004	-3.8611
f_{20}	-3.32	7172.5	4.44E-004	-3.3205	3933	4.05E-004	-3.3204
f_{21}	-10	7035.6	3.97E-002	-9.98	5792.7	3.96E-002	-9.54
f_{22}	-10	6649	9.89E-002	-9.99	4917.8	9.50E-002	-9.78
f_{23}	-10	6846.6	1.25E-001	-9.97	4865.2	1.25E-001	-9.69

Tabla 4.1: Comparación de ED y NSSDE

de evaluaciones de la función objetivo. Por el contrario, si el punto inicial de búsqueda está muy alejado de un óptimo local y, tomando en cuenta la alta no linealidad de las funciones de prueba adoptadas, la aplicación del algoritmo de Nelder-Mead resulta poco eficiente en estas circunstancias.

En términos generales, nuestro algoritmo híbrido NSSDE, resultó ser muy competitivo y consistente sobre el conjunto de funciones del prueba de Yao et al. [83]. La definición de estas funciones de prueba, se presenta en el apéndice A.

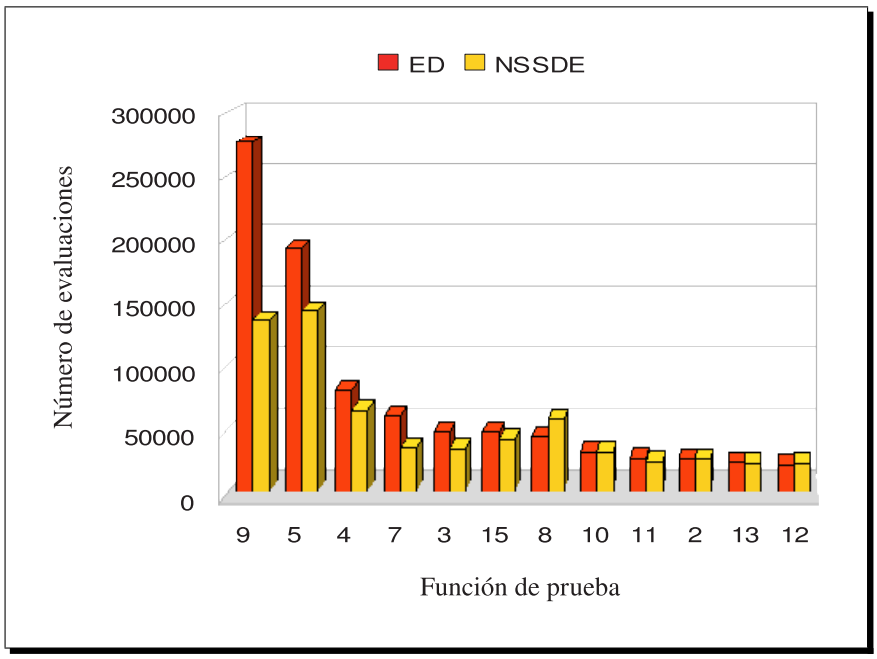

Figura 4.4: Gráfica comparativa entre ED y NSSDE (parte 1)

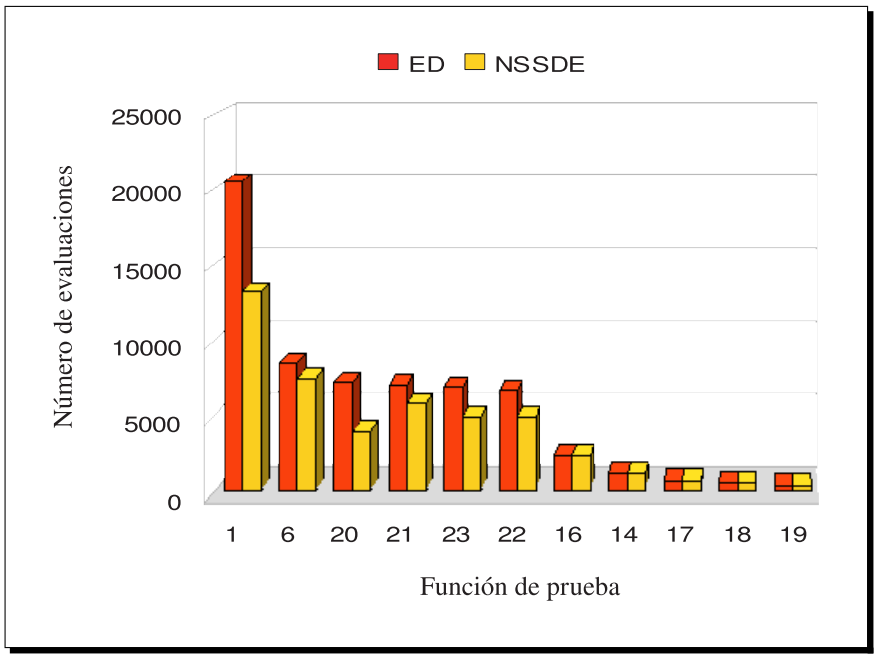

Figura 4.5: Gráfica comparativa entre ED y NSSDE (parte 2)

CAPÍTULO 5

PROPUESTA MULTIOBJETIVO

En el capítulo anterior, describimos una estrategia para acoplar a un algoritmo evolutivo el método de Nelder-Mead, el cual actúa como buscador local. Para el caso de optimización mono-objetivo, nuestra estrategia es retomar la misma técnica y emplearla en este nuevo algoritmo híbrido.

En el resto del capítulo, describiremos nuestra propuesta algorítmica denominada *Nonlinear Simplex Search Genetic Algorithm* (NSS-GA) para optimización multiobjetivo. Nuestro algoritmo está basado en el NSGA-II [15], utilizando los algoritmos de optimización matemática de Nelder-Mead [53] (para funciones multidimensionales) y el de la sección dorada (para funciones unidimensionales) como estrategias de búsqueda local. También presentaremos los trabajos relacionados a algoritmo híbridos multiobjetivo, que utilizan métodos de programación matemática en su hibridización. Además, describiremos los problemas que se presentaron en los experimentos realizados, así como las estrategias tomadas para solucionarlos. Adicionalmente, presentaremos las métricas utilizadas para la evaluación de nuestro algoritmo. Y por último, expondremos los resultados de nuestra propuesta y las conclusiones sobre los resultados obtenidos por nuestro algoritmo híbrido.

5.1. Trabajos relacionados

El interés en el diseño e implementación de meta-heurísticas híbridas se ha incrementado de forma importante en los últimos años [73]. Sin embargo, la mayoría de estos estudios se han centrado en técnicas mono-objetivo, mientras que el ámbito multiobjetivo constituye aún, una área poco explorada de investigación.

En la actualidad, se han desarrollado algoritmos híbridos multiobjetivo como los presentados en [3, 33, 36, 39]. En algunos casos, han sido desarrollados para aplicaciones muy específicas e hibridizados con propósitos distintos a las de esta tesis. Nosotros abordamos el problema de optimización multiobjetivo no lineal, utilizando como algoritmo

explorativo al NSGA-II y el método de Nelder-Mead como buscador local.

Son pocos los trabajos que utilizan métodos clásicos de optimización, como buscadores locales en optimización de multiobjetivo. A continuación presentamos los dos trabajos que más se asemejan a nuestra propuesta, en el sentido de utilizar métodos clásicos de optimización en su hibridización.

- En 2003, Hu et al. [32] combinan la programación cuadrática secuencial (*Sequential Quadratic Programming, SQP*) y el método de la restricción ε con los algoritmos evolutivos multiobjetivo SPEA y NSGA-II. El potencial que enmarca este algoritmo híbrido, radica en aprovechar la propiedad de convertir un POM a un problema de optimización mono-objetivo (estrategia natural del método de la restricción ε). Para cada función se le aplica una búsqueda local utilizando SQP. Esta hibridización permite una mejora en la calidad de las soluciones en diferentes funciones de prueba. Además, con esta estrategia híbrida se permite incrementar la diversidad de las soluciones obtenidas. Los detalles del este algoritmo se describen en [32].

- Recientemente (en 2007), Koduru et al. [41] proponen una hibridización basada en el algoritmo de PSO y el método simplex no lineal. La idea para acoplar el método de Nelder-Mead es similar a un trabajo realizado anteriormente por los mismos autores [40] (descrito en el capítulo anterior). Este algoritmo híbrido multiobjetivo se basa en la dominancia difusa (*fuzzy dominance*) [42] y utiliza un archivo externo que contiene las soluciones difusas no dominadas. En términos generales, la hibridización consiste en definir k cúmulos (*clusters*) de la población con cardinalidad de $n+1$ en el espacio de las funciones objetivo (donde n es la dimensión de la función). Estos cúmulos representan los vértices del simplex. El método de Nelder-Mead se aplica a cada cúmulo definido y se repite por un determinado número de iteraciones. Se actualiza el archivo externo con las partículas de los cúmulos de acuerdo a la dominancia difusa y se continúa con la siguiente iteración.

5.2. Nuestra propuesta

De la misma manera que la propuesta anterior, el mecanismo de aprendizaje adoptado en nuestro algoritmo híbrido multiobjetivo denominado *Nonlinear Simplex Search Genetic Algorithm* (NSS-GA), se basa en la teoría de la evolución Lamarckiana.

Hemos adoptado el algoritmo NSGA-II como método base de exploración, además el método de Nelder-Mead y el de la sección dorada en la fase de explotación. A continuación presentaremos las dos fases que componen nuestra propuesta y mencionaremos los problemas que se presentaron. También describiremos las estrategias adoptadas para solucionar dichos problemas.

5.2.1. Fase de exploración

Este mecanismo tiene como objetivo generar diversas soluciones en distintas zonas del espacio de búsqueda. El método que utilizamos en esta fase, es el algoritmo NSGA-II descrito en el capítulo 3. De esta manera, aprovechamos el proceso evolutivo del algoritmo genético para diversificar los individuos de la población, generación tras generación.

5.2.2. Fase de explotación

El método de Nelder-Mead, es un método directo de optimización utilizado para optimizar funciones multi-dimensionales en espacios continuos. En cuanto a funciones unidimensionales se refiere, se ha demostrado que el método de Nelder-Mead converge únicamente en funciones estrictamente convexas [43]. Es por esto, que introdujimos el método de la sección dorada, el cual, a diferencia del método de Nelder-Mead, requiere una única evaluación de la función objetivo por cada iteración.

En el capítulo anterior presentamos el criterio de parada de la búsqueda local el cual retomamos en este algoritmo multiobjetivo. Para el caso particular de la sección dorada, el criterio de parada cambia. Nosotros proponemos un valor de $\varepsilon = 1.0 \times 10^{-3}$ como criterio de convergencia del algoritmo de la sección dorada (ver algoritmo 1, capítulo 2).

En nuestra propuesta, realizamos una búsqueda local para cada una de las funciones del problema multiobjetivo. Además, incluimos una función agregativa con la finalidad de obtener mejores soluciones compromiso, entre las soluciones que optimizan a cada función del problema multiobjetivo. La idea es intensificar la búsqueda hacia una mejor solución para cada función, partiendo de un individuo de la población. De esta manera, los operadores genéticos del algoritmo evolutivo, reproducirán los rasgos genéticos de las mejores soluciones por función y la selección natural elegirá las soluciones más aptas para sobrevivir en el ambiente (el problema multiobjetivo).

Desafortunadamente, la alta dimensionalidad y multimodalidad de las funciones son el gran problema que puede volver ineficientes a los métodos clásicos de optimización. Un ejemplo claro recae en el método de Nelder-Mead, el cual resulta inoperable para determinadas clases de funciones, como por ejemplo la función de McKinnon [48], en donde el mismo autor demuestra la convergencia del método hacia un punto no estacionario (o no óptimo).

Con base en los estudios experimentales realizados, se detectó que aplicar el método de Nelder-Mead en un simplex regular, puede resultar completamente ineficaz para optimizar una determinada función. Un ejemplo claro son los problemas del conjunto denominado ZDT [85]. Aunque en la literatura especializada existen trabajos para agilizar la convergencia del método de Nelder-Mead hacia óptimos locales, como el simplex difuso (*fuzzy simplex*) de Trabia [75], los movimientos inteligentes de Rahman [58] y otras modificaciones que se han hecho al método [4], en realidad no se ha logrado generalizar

la técnica para solucionar el problema cuando el simplex inicial resulta inoperable para optimizar una determinada función.

Con lo descrito anteriormente, surgen nuevas interrogantes, tales como: 1) ¿Qué individuo seleccionar para aplicar la búsqueda local?, 2) ¿Cómo construir el simplex? y 3) ¿Cómo manejar las restricciones del límite de las variables?. Las estrategias tomadas para satisfacer estas nuevas inquietudes se presentan a continuación.

5.2.2.1. Selección de individuo para aplicar búsqueda local

Dada una población P, seleccionamos al individuo $\vec{x}_\Delta \in P$ para optimizar con respecto del objetivo i de acuerdo a:

$$\vec{x}_\Delta = \vec{x}_l | \vec{x}_l = \min_{\forall \vec{x}_l \in \mathcal{P}^*} \{ f_k(\vec{x}_l) \} \tag{5.1}$$

donde $\mathcal{P}^*$ es el conjunto de soluciones no dominadas en la población P.

En otras palabras, seleccionaremos al individuo con mejor aptitud de acuerdo a la i-ésima función del problema multiobjetivo y que está en el conjunto de soluciones no dominadas de la población actual P.

5.2.2.2. Construcción del simplex

A la solución x_Δ seleccionada, la denominamos "cabeza del simplex" y será el primer vértice del *n-simplex* (donde n es la dimensión de la función), esto es:

$$\Delta_0 = x_\Delta$$

Los n vértices restantes, son elegidos de tal manera que se forme un conjunto linealmente independiente. De manera similar que en [9], nosotros empleamos una estrategia para reducir el dominio de la búsqueda local. En estas condiciones, los vértices son construidos dentro de este nuevo dominio. A continuación presentamos la estrategia que proponemos para la reducción del dominio de búsqueda y la manera en que construimos los puntos en este nuevo dominio.

Reducción del dominio de búsqueda

A diferencia del trabajo de Chelouah y Siarry [9], nosotros no requerimos de un parámetro adicional para reducir el dominio de búsqueda. La idea que proponemos, es realizar un análisis en la estructura genética de una muestra de la población, que es tomada de acuerdo con los mejores individuos con respecto a la i-ésima función a optimizar. Con esta muestra poblacional, podemos identificar la media aritmética y la desviación estándar de los genes (parámetros) de cada individuo de la muestra. Entonces, el espacio prometedor para el óptimo de la i-ésima función estará acotado por un nuevo límite inferior ($low\_bound^j_{new}$)

y superior ($up\_bound^j_{new}$) con respecto del j-ésimo parámetro, respectivamente. Definimos los nuevos límites de la búsqueda local, de acuerdo a:

$$\begin{aligned} low\_bound^j_{new} &= m(P_m(j)) - \sigma(P_m(j)) \\ up\_bound^j_{new} &= m(P_m(j)) + \sigma(P_m(j)) \end{aligned} \tag{5.2}$$

donde P_m es la muestra poblacional obtenida, $m(P_m(j))$ la media aritmética y $\sigma(P_(j))$ la desviación estándar del j-ésimo parámetro de la muestra P_m.

Para este trabajo, la muestra poblacional es del 20 % con respecto al tamaño original de la población P, pero puede variar dependiendo de la función a optimizar.

Construcción de los vértices del simplex

Una vez obtenido el nuevo dominio de búsqueda, el resto de los vértices del simplex son creados de acuerdo a una buena distribución en dicho dominio.

Las secuencias de baja discrepancia (*low-discrepancy sequence*), son denominadas también secuencias cuasi-aleatorias o sub-aleatorias y ofrecen un grado de uniformidad más alto (de baja discrepancia), con respecto a la distribución uniforme de números pseudo-aleatorios común.

Para obtener los n vértices restantes del simplex, proponemos la construcción de los puntos de acuerdo a dos secuencias de baja discrepancia, que no son más que la generalización de la secuencia de van der Corput [77]. Estas secuencias tienen sus fundamentos en la teoría de números y se presentan a continuación.

La primera de ellas conociada como secuencia de Halton (*Halton sequence*) [27], difiere de la representación binaria de la secuencia de van der Corput, debido a que utiliza una base diferente para cada coordenada del vector. De esta manera, la discrepancia resultante y dispersión es mejor que una simple muestra con distribución uniforme. Supongamos que queremos construir el i-ésimo vector de la secuencia de Halton en $\mathbb{R}^n$. El primer paso es elegir a n primos relativos $p_1, p_2, \ldots, p_n$ (comúnmente los primeros n números primos $p_1 = 2, p_2 = 3, \ldots$). Para construir el i-ésimo vector de la secuencia, se considera la representación en base p (número primo elegido) de i, la cual toma la forma de $i = a_0 + pa_1 + p^2a_2 + \ldots$. Cada coordenada del vector estará en el intervalo $[0,1]$ y se obtiene invirtiendo el orden de los bits, esto es:

$$r(i,p) = \frac{a_0}{p} + \frac{a_1}{p^2} + \frac{a_2}{p^3} + \frac{a_3}{p^4} + \ldots \tag{5.3}$$

De esta manera, el i-ésimo vector en la secuencia de Halton (iniciando con $i = 0$) está dado por:

$$\langle r(i,p_1), r(i,p_2), \ldots, r(i,p_n) \rangle \tag{5.4}$$

Algoritmo 13: Secuencia de Halton

Input: i (i-ésimo vector), j (j-ésimo componente del vector i)
Output: $c \in [1,0]$ (componente j del vector i) en la secuencia de Halton

```
1  begin
2      prime[n] = [2,3,5,7,...];        /* definir los primeros n números primos */
3      p1 = prime[j];
4      p2 = p1;
5      c = 0;
6      repeat
7          x = i  mod p1;
8          c = c + x/p2;
9          i = ⌊i/p1⌋;
10         p2 = p2 * p1;
11     until (i ≤ 0) ;
12     return c;
13 end
```

Supongamos que conocemos el número total de vectores k que se deseen crear. En este caso, se puede tener una mejor distribución de puntos. La segunda distribución de puntos denominada secuencia de Hammersley (*Hammersley sequence*) [28], es una adaptación de la secuencia de Halton. Utiliza únicamente $n-1$ números primos, iniciando con $i = 0$, el i-ésimo vector en la secuencia de Hammersley para un conjunto total de k vectores está dado por:

$$\left\langle \frac{1}{k}, r(i,p_1), r(i,p_2), \ldots, r(i,p_{n-1}) \right\rangle \tag{5.5}$$

El algoritmo 13, muestra el pseudo-código para obtener el j-ésimo componente del vector i de la secuencia de Halton. El valor c que regresa el algoritmo está en el intervalo $[0,1]$ y puede ser mapeado a un valor c_{map} en el intervalo $[low, up]$ de acuerdo a:

$$c_{map} = low + c \cdot (up - low)$$

El diagrama de Voronoi [80, 81] es una estructura geométrica que representa información de proximidad con respecto de un conjunto de puntos u objetos. En otras palabras, es una partición del espacio en celdas, cada una de la cuales contiene una colindancia con sus puntos más cercanos. En la figura 5.1, presentamos el diagrama de Voronoi de un conjunto de doscientos puntos generados a partir de distintas distribuciones.

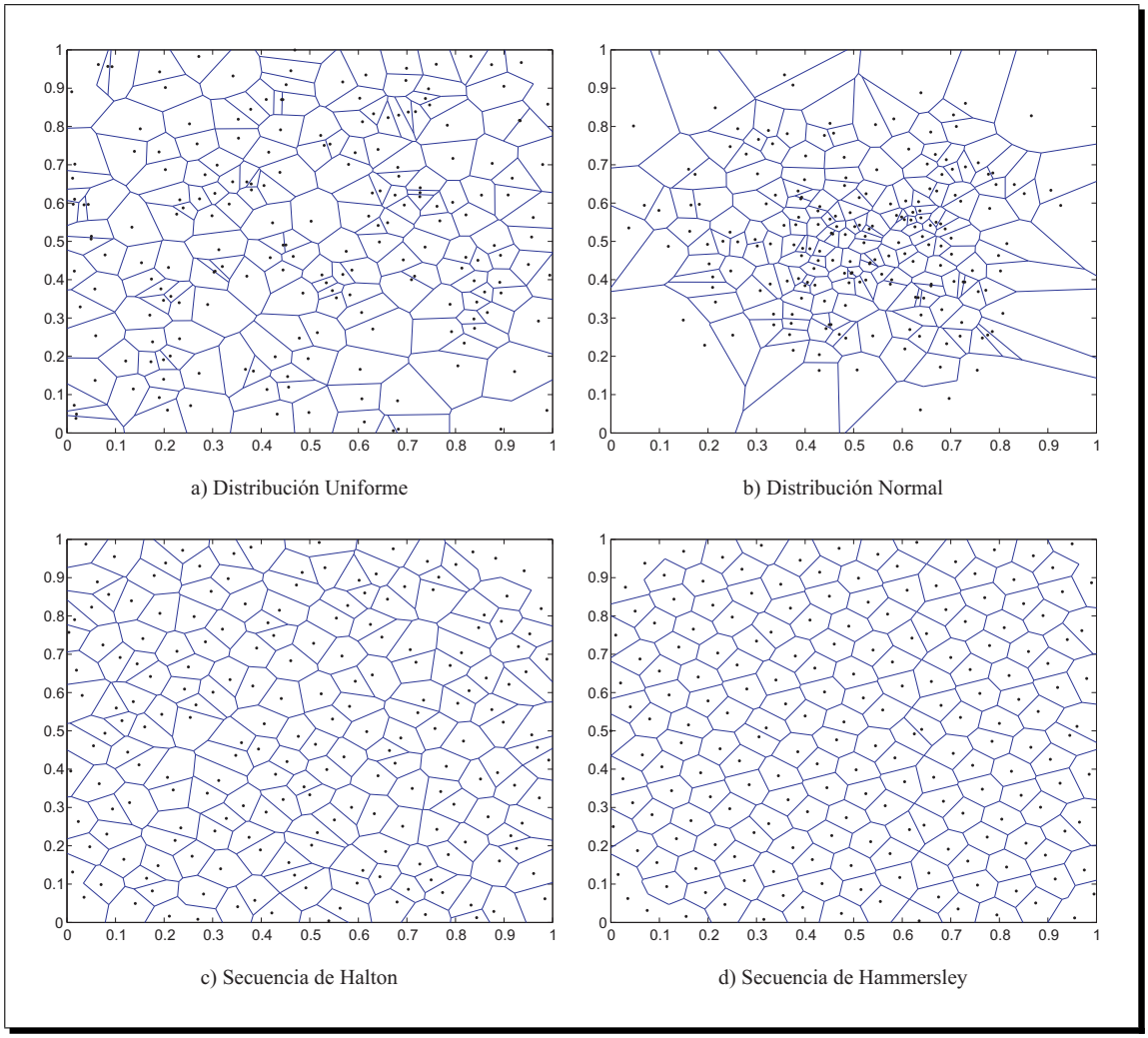

Figura 5.1: Doscientos puntos creados con: a) una distribución uniforme; b) una distribución normal, con media .5 y desviación estándar .5; c) la secuencia de Halton en $\mathbb{R}^2$ y d) la secuencia de Hammersley en $\mathbb{R}^2$

Como se observa, la secuencia de Hammersley (figura 5.1.d), tiene una mejor distribución de puntos con respecto de la distribución uniforme (figura 5.1.a), la distribución normal con media .5 y desviación estándar .5 (figura 5.1.b) e incluso que la secuencia de Halton (figura 5.1.c).

En nuestros experimentos realizados, notamos que las secuencias de Halton y Hammersley da mejores resultados con respecto de las otras dos distribuciones. Es por esto que descartamos la distribución uniforme y normal, para generar los vértices restantes del simplex. Para generar el resto de los vértices en el nuevo dominio, proponemos utilizar la secuencia de Halton o Hammersley seleccionándola de acuerdo a una probabilidad para cada invocación de la construcción del simplex.

5.2.2.3. Límites de las variables

Como se ha mencionado, por cada iteración del método de Nelder-mead se genera un nuevo simplex, el cual es no degenerado siempre y cuando el simplex inicial no lo sea.

Desafortunadamente, los movimientos que realiza Nelder-Mead en el simplex, pueden ocasionar que el nuevo vértice del simplex esté fuera del rango de las variables. En [44], se propone una estrategia que es tomada en este trabajo y se presenta a continuación.

Sea Δ_i el nuevo vértice creado por algún movimiento del algoritmo de Nelder-Mead. Para evitar que algún componente del vértice Δ_i sobrepase los límites permitidos del j-ésimo parámetro, éste será restablecido de acuerdo a la ecuación 5.6.

$$\Delta_i^{(j)} = \begin{cases} low\_bound_j & \text{, si } \Delta_i^{(j)} < low\_bound_j \\ up\_bound_j & \text{, si } \Delta_i^{(j)} > up\_bound_j \\ \Delta_i^{(j)} & \text{, en otro caso.} \end{cases} \tag{5.6}$$

donde low_bound_j y ip_bound_j son el límite inferior y superior, respectivamente.

Tomar esta alternativa provoca un trastorno en los movimientos naturales del algoritmo de Nelder-Mead, pudiendo ocasionar que el simplex se degenere y, en consecuencia, disminuya la dimensión de la búsqueda.

Nosotros proponemos verificar si el simplex no se ha degenerado cuando se invoca a la restauración de parámetros. En caso de haberse degenerado, la búsqueda local termina y se puede generar otro simplex para una nueva búsqueda local.

Un simplex es no degenerado siempre y cuando su volumen sea mayor que cero. Podemos calcular el volumen del simplex de acuerdo a lo siguiente:

Sea Δ_k la matriz de $n \times (n+1)$ la cual representa el simplex Δ, esto es:

$$\Delta_k = \begin{pmatrix} \Delta_0^{(1)} & \Delta_1^{(1)} & \cdots & \Delta_n^{(1)} \\ \Delta_0^{(2)} & \Delta_1^{(2)} & \cdots & \Delta_n^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ \Delta_0^{(n)} & \Delta_1^{(n)} & \cdots & \Delta_n^{(n)} \end{pmatrix}$$

es decir, cada columna representa el i-ésimo vértice del simplex. Entonces el volumen del simplex puede ser calculado de acuerdo a [43]:

$$vol(\Delta) = \frac{|det(M_k)|}{n!} \tag{5.7}$$

donde M_k es una matriz de tamaño $n \times n$ definida como:

$$M_k = \begin{pmatrix} \Delta_0^{(1)} - \Delta_n^{(1)} & \Delta_1^{(1)} - \Delta_n^{(1)} & \cdots & \Delta_{n-1}^{(1)} - \Delta_n^{(1)} \\ \Delta_0^{(2)} - \Delta_n^{(2)} & \Delta_1^{(2)} - \Delta_n^{(2)} & \cdots & \Delta_{n-1}^{(2)} - \Delta_n^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ \Delta_0^{(n)} - \Delta_n^{(n)} & \Delta_1^{(n)} - \Delta_n^{(n)} & \cdots & \Delta_{n-1}^{(n)} - \Delta_n^{(n)} \end{pmatrix}$$

En caso de que $vol(\Delta) = 0$, entonces el simplex se ha degenerado y aunque en la práctica no es muy común que se degenere, es recomendable tener en cuenta este criterio.

5.2.2.4. Función agregativa

Una vez que se ha optimizado cada función del problema multiobjetivo, nosotros proponemos minimizar una función agregativa; y para este enfoque, la selección del individuo del cual parte la búsqueda local cambia. Proponemos seleccionar al individuo cuya aptitud revele el mejor compromiso entre las soluciones obtenidas, que minimizan cada función del problema multiobjetivo. La selección de esta solución se describe a continuación.

Dado el vector $\vec{F} = [f_1^*, f_2^*, \ldots, f_k^*]$, cuyos componentes son los valores mínimos f_i^* de las k funciones objetivo encontradas en la generación actual. Seleccionamos al individuo $\vec{x}_i$ de la población P, tal que minimice la función:

$$G(\vec{x}) = \sum_{i=1}^{k} \frac{|F[i] - f_i(\vec{x})|}{|F[i]|} \tag{5.8}$$

y a partir de $\vec{x}$ se realiza la búsqueda local, minimizando la función agregativa descrita a continuación.

En un problema multiobjetivo se desean tener las soluciones que se encuentran más cercanas al verdadero frente de Pareto. Y para los problemas de minimización, estas soluciones en teoría estarán más cerca del vector ideal artificial $\vec{F}$. La función agregativa, que presentamos se basa en este hecho. En este enfoque, la función que se optimiza mediante el método de mejora, está dada por la ecuación 5.9.

$$\psi(\vec{x}) = ED(\vec{F}, \vec{H}(\vec{x})) \tag{5.9}$$

donde $\vec{H}$ es el vector de funciones objetivo en el punto $\vec{x}$ y ED es la distancia euclidiana (*Euclidean Distance*) entre los vectores $\vec{F}$ y $\vec{H}$, calculada por:

$$ED(F,H) = \sqrt{\sum_{i=1}^{k} (F[i] - H[i])^2}$$

Resumiendo, la fase de explotación del algoritmo optimiza cada una de las funciones del problema multiobjetivo así como la función agregativa propuesta. Los pasos que se llevan a cabo en esta fase son:

1. Seleccionar al individuo del cual partirá la búsqueda local, esto con base en la ecuación 5.1 para las funciones del problema multiobjetivo y la ecuación 5.8 para la función agregativa.

2. Aplicar la búsqueda local con respecto a las k funciones del problema multiobjetivo; y después, a la función agregativa.

   - Si la función es unidimensional:

     a) Utilizar el método de la sección dorada.

   - Si la función es multidimensional:

     a) Reducir el dominio de búsqueda.

     b) Construir el simplex en el nuevo dominio.

     c) Aplicar el método de Nelder-Mead optimizando cada función, hasta que el criterio de parada de la ecuación 4.4 (capítulo 4) se satisfaga.

Una vez explicada la fase explotativa del algoritmo híbrido, ahora describiremos la mecánica completa de nuestro algoritmo.

5.2.3. El algoritmo

Inicialmente, se crea una población aleatoria P_0 de cardinalidad N. Al principio del algoritmo, se aplican los operadores de selección, cruza y mutación para obtener una población de descendientes Q_0 de tamaño N. De esta manera se obtiene una población $R_t = P_0 \cup Q_0$ de tamaño $2N$. Después, la población R_t se ordena de acuerdo a los principios de no dominación y se obtiene la distancia de agrupamiento de cada individuo. Para la siguiente generación P_{t+1} se seleccionan los N individuos de R_t de acuerdo al operador de comparación de agrupamiento $\prec_n$ (ver pag. 35, capítulo 3) y se continúa con la siguiente iteración.

Con base en nuestros estudios experimentales, proponemos aplicar la fase de mejora con una frecuencia mayor que en la propuesta mono-objetivo. Nuestra propuesta es aplicarlo sobre la población P_i cada $\frac{n}{2}$ generaciones, donde n es la dimensión de la función. Dicha frecuencia depende directamente del tamaño de dimensión de la función y puede variar de acuerdo al problema multiobjetivo.

En la fase de mejora, se obtienen soluciones localmente óptimas tanto para cada función del problema multiobjetivo, como para la función agregativa. Para el caso particular del método de la sección dorada, se tendrá una única solución, mientras que para el método de Nelder-Mead, se tendrá un conjunto de soluciones representadas por el simplex final.

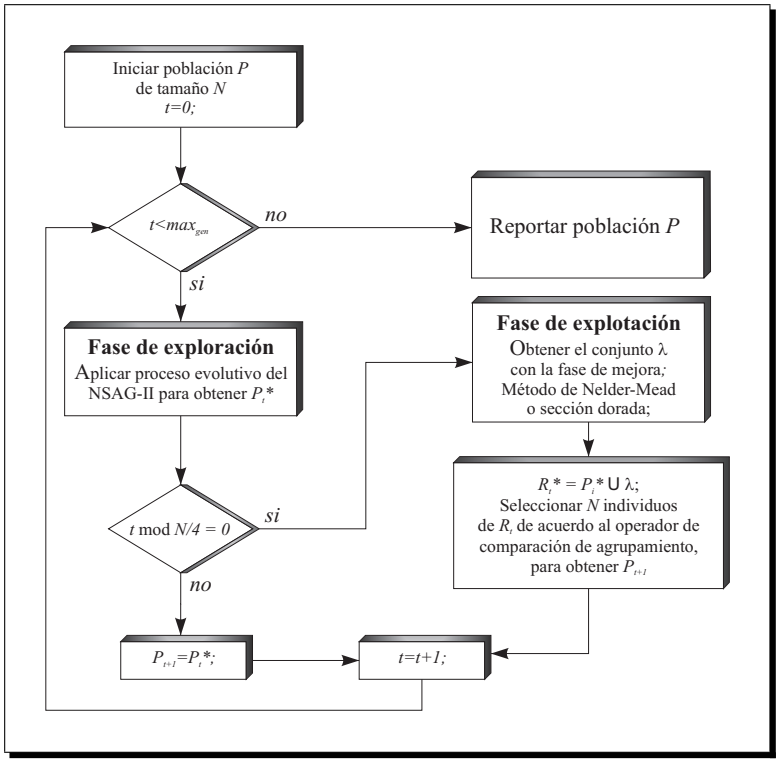

Figura 5.2: Esquema del algoritmo híbrido NSS-GA

La unión de todas las soluciones localmente óptimas, denotado por λ, representa el conjunto de óptimos locales y serán puestos en competencia con la población P_i para ser o no seleccionados por el operador de comparación de agrupamiento. La explicación del algoritmo puede verse gráficamente en la figura 5.2, mientras que el algoritmo 14 muestra el pseudo-código del NSS-GA.

Algoritmo 14: Nonlinear Simplex Search Genetic Algorithm (NSS-GA)

Input: Un número máximo de generaciones max_{gen}
Output: Una población evolucionada P

begin
  $t = 0$;
  Inicializar una población aleatoria P_t de tamaño N;
  Evaluar la aptitud de acuerdo a las funciones objetivo;
  while $(t < max_{gen})$ **do**
    Aplicar cruza y mutación a P_t para generar Q_t;
    Evaluar población Q_t de acuerdo a las funciones objetivo;
    $R_t = P_t \cup Q_t$;
    Asignar jerarquía con base en la dominancia de Pareto a R_t;
    Asignar distancia de a grupamiento a R_t;
    Seleccionar N individuos de R_t de acuerdo al operador de comparación de agrupamiento $\prec_n$, para obtener P_t^*;
    if $((t \mod \frac{N}{4}) = 0)$ **then**
      Iniciar la fase de mejora para obtener el conjunto de soluciones λ;
      $R_t^* = P_t^* \cup \lambda$;
      Asignar jerarquía con base en la dominancia de Pareto a R_t^*;
      Asignar distancia de a agrupamiento a R_t^*;
      Seleccionar N individuos de R_t^* de acuerdo al operador de comparación de agrupamiento $\prec_n$, para obtener P_{t+1};
    else
      $P_{t+1} = P_t^*$;
    end
    $t = t + 1$;
  end
end

5.3. Métricas

Con el objetivo de poder realizar una comparación entre nuestro algoritmo híbrido y el algoritmo evolutivo base (el NSGA-II), hacemos uso de las métricas de distancia generacional $\mathcal{DGI}$, espaciamiento $\mathcal{E}$ y cobertura $\mathcal{C}$.

5.3.1. Distancia Generacional Invertida ($\mathcal{DGI}$)

La distancia generacional ($\mathcal{DG}$) propuesta por Van Veldhuizen y Lamont [78, 79] es una métrica que indica qué tan lejos o cerca están los puntos obtenidos por un algoritmo multiobjetivo del frente de Pareto verdadero.

Dado el verdadero frente de Pareto Γ, la distancia generacional para un conjunto de soluciones P está determinada por:

$$\mathcal{DG} = \frac{\sqrt{\sum_{i=1}^{|P|} d_i^2}}{|P|} \tag{5.10}$$

donde d_i es la distancia euclidiana entre la i-ésima solución de P y la solución más cercana a Γ, es decir:

$$d_i = \min_{j=1}^{|\Gamma|} \sqrt{\sum_{k=1}^{M} (f_k(i) - f_k(j))^2}$$

donde M es el número de funciones objetivo.

La $\mathcal{DGI}$ utiliza el mismo concepto. Si embargo, a diferencia de la $\mathcal{DG}$, se utiliza como base el verdadero frente de Pareto Γ. Un valor de $\mathcal{DGI}$ igual a cero significa que todas las soluciones obtenidas por el algoritmo pertenecen al conjunto de óptimos de Pareto.

5.3.2. Espaciamiento ($\mathcal{E}$)

La métrica de espaciamiento propuesta por Schott [66] cuantifica la dispersión de las soluciones en el frente no dominado. Evalúa la varianza de las distancias más cercanas entre el vecindario de soluciones. Dado un conjunto P de soluciones, la métrica puede ser calculada como:

$$\mathcal{E} = \sqrt{\frac{1}{|P| - 1} \sum_{i=1}^{|P|} (\overline{d} - d_i)^2} \tag{5.11}$$

donde d_i y $\overline{d}$ son definidas como:

$$d_i = \min_{i,i\neq j} \left\{ \sum_{k=1}^{M} |f_k^i - f_k^j| \right\}$$

$$\overline{d} = \frac{\sum_{i=1}^{|P|} d_i}{|P|}$$

Un valor bajo de $\mathcal{E}$ indica una mejor distribución de las soluciones en el frente y un valor igual a cero alude que todas las soluciones están uniformemente distribuidas.

5.3.3. Cobertura ($\mathcal{C}$)

Propuesta por Zitzler et al. en [86]. Esta métrica compara un conjunto de soluciones con respecto a otro, utilizando la dominancia de Pareto. la métrica se define como:

$$\mathcal{C}(A,B) = \frac{|\{b \in B | \exists a \in A : a \preceq b\}|}{|B|} \tag{5.12}$$

Si todos los puntos en A dominan o son iguales a todos los puntos de B, esto implica que $\mathcal{C}(A,B) = 1$. Si ningún punto de A domina a algún punto del conjunto B entonces $\mathcal{C}(A,B) = 0$. Cuando $\mathcal{C}(A,B) = 1$ y $C(B,A) = 0$ entonces se dice que el conjunto A es mejor que B. Debido a que el operador de dominancia no es simétrico, se necesita calcular $\mathcal{C}(A,B)$ y $\mathcal{C}(B,A)$ para comprender cuántas soluciones de A son cubiertas por B y viceversa.

5.4. Resultados obtenidos

El objetivo principal en esta tesis, fue diseñar un algoritmo híbrido que convergiera al frente de Pareto verdadero, reduciendo el número de evaluaciones de la función de aptitud. La estrategia adoptada y consistió en acoplar un operador de búsqueda local a un algoritmo evolutivo multiobjetivo, del estado del arte (el NSGA-II).

Para las pruebas realizadas se fijó un número máximo de evaluaciones de la función objetivo y se evaluó el desempeño tanto del algoritmo híbrido como del algoritmo evolutivo adoptado. Recordando que nuestro algoritmo no necesita algún parámetro adicional con respecto del algoritmo evolutivo base adoptado, las comparaciones se realizaron con los mismos parámetros en ambos algoritmos para una justa competencia. Dichos parámetros se muestran en la tabla 5.1.

Parámetro	NSS-GA	NSGA-II
T_p	100	100
P_c	.9	.9
P_m	$\frac{1}{N}$	$\frac{1}{N}$

Tabla 5.1: Parámetros utilizados para la comparación entre el NSS-GA y el NSGA-II

En la tabla 5.1, T_p es el tamaño de la población, P_c es la probabilidad de cruza, P_m es la probabilidad de mutación y N representa el número de variables del problema multiobjetivo.

En resumen, el proceso de comparación se llevó a cabo, realizando 30 ejecuciones de los algoritmos para cada problema multiobjetivo. Las ejecuciones se restringen a 4,000 evaluaciones de la función de aptitud. Por cada función se probaron las tres métricas presentadas anteriormente ($\mathcal{GDI}$, $\mathcal{E}$ y $\mathcal{C}$). Cabe reiterar que un valor cercano a 0 para la métrica $\mathcal{DGI}$ indica un mejor acercamiento al verdadero frente de Pareto; un valor cercano de 0 para la métrica $\mathcal{E}$ indica una mejor dispersión en las soluciones; y para el caso de la cobertura, un valor cerca a 1 indica que el algoritmo domina a la mayoría de las soluciones de su contrincante; por el contrario, un valor igual a 0 indica que todas las soluciones son dominadas.

Cada tabla comparativa muestra los valores de la media y desviación estándar σ de los resultados de las métricas en las 30 ejecuciones realizadas para cada algoritmo. Además, para tener una clara visión de la comparación de los algoritmos, se muestran las gráficas correspondientes a la media con respecto de la métrica $\mathcal{GDI}$. En las pruebas realizadas, se puede observar que no se llega al frente de Pareto verdadero. Sin embargo, si se deja correr el algoritmo por un mayor número de generaciones, nuestra propuesta algorítmica convergerá al verdadero frente, tal y como se muestra en las gráficas del apéndice C.

A continuación se muestran los resultados al evaluar cinco problemas multiobjetivo propuestos por Zitzler [85] y dos más, propuestos por Deb [17] (ver apéndice B).

5.4.1. ZDT1

Para esta función de prueba, la tabla 5.2, muestra claramente que el algoritmo NSS-GA, obtiene mejores resultados en las métricas $\mathcal{GDI}$, $\mathcal{E}$ y $\mathcal{C}$. Y aunque la estabilidad del NSGA-II es mejor, las diferencias no son realmente significativas. Para respaldar los resultados de las métricas, en la figura 5.3 se observa que nuestra propuesta tiene una mejor aproximación al verdadero frente. Más aún, todas las soluciones de nuestro algoritmo dominan a las soluciones del NSGA-II.

ZDT1

Métrica	NSS-GA		NSGA-II	
	media	σ	*media*	σ
$\mathcal{DGI}$	0.001149	0.000598	0.005582	0.000905
$\mathcal{E}$	0.014620	0.005329	0.023731	0.004730
$\mathcal{C}$	1.000000	0.000000	0.000000	0.000000

Tabla 5.2: Resultado de las métricas para la función ZDT1

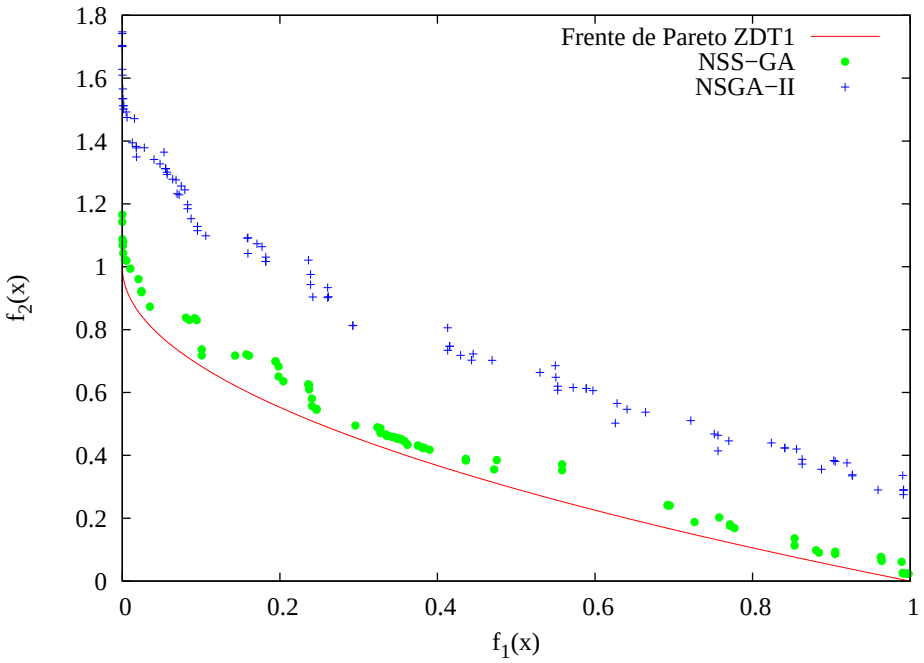

Figura 5.3: Comparación del NSS-GA y el NSGA-II en el problema multiobjetivo ZDT1

5.4.2. ZDT2

En el problema ZDT2 ocurre algo similar: en todas las métricas, el algorimo NSS-GA supera al NSGA-II, tal y como lo revela la tabla 5.3. Es claro ver en la figura 5.4, que nuestro algoritmo casi toca el frente de Pareto verdadero, mientras que el algoritmo del NSGA-II está todavía bastante alejado del frente verdadero.

ZDT2				
Métrica	NSS-GA		NSGA-II	
	media	σ	*media*	σ
$\mathcal{DGI}$	0.002101	0.001785	0.015385	0.004631
$\mathcal{E}$	0.021928	0.014674	0.029762	0.006576
$\mathcal{C}$	0.971111	0.155571	0.004444	0.023934

Tabla 5.3: Resultado de las métricas para la función ZDT2

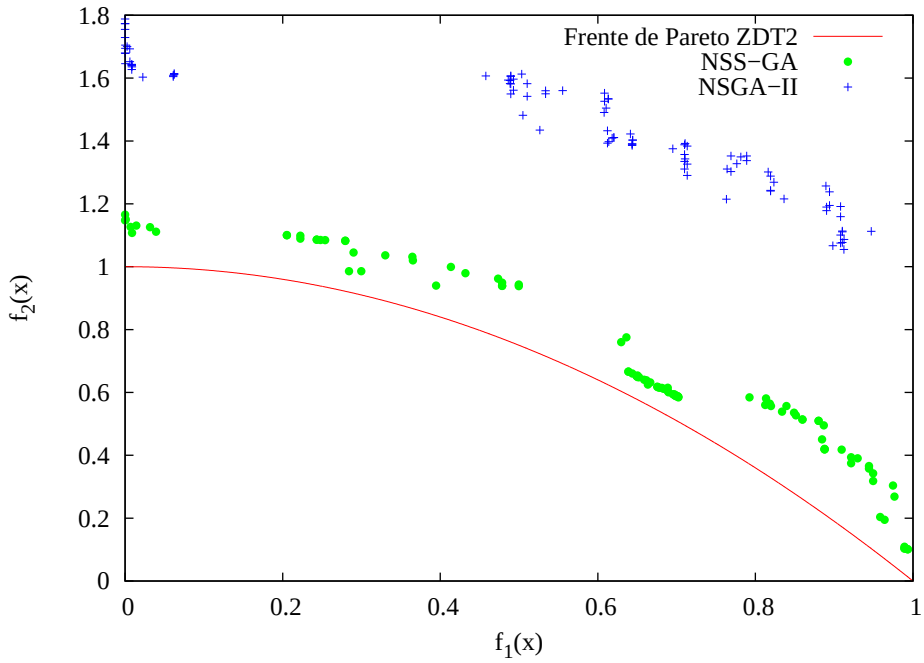

Figura 5.4: Comparación del NSS-GA y el NSGA-II en el problema multiobjetivo ZDT2

5.4.3. ZDT3

Este problema multiobjetivo, se destaca por tener un frente desconectado. En la figura 5.5 se muestra claramente que el NSS-GA obtiene un mejor acercamiento al verdadero frente de Pareto en comparación con el NSGA-II. Incluso, con 4,000 evaluaciones se llega a tocar el verdadero frente. La tabla 5.4 muestra que en todas las métricas, nuestra propuesta es mejor que el algoritmo NSGA-II. Aunque la desviación estándar revela que la estabilidad del NSGA-II es mejor, no significa que nuestro algoritmo sea inestable ya que la diferencia es poco significativa.

ZDT3

Métrica	NSS-GA		NSGA-II	
	media	σ	*media*	σ
$\mathcal{DGI}$	0.001221	0.000832	0.004217	0.000798
$\mathcal{E}$	0.013990	0.005108	0.023994	0.004774
$\mathcal{C}$	0.969534	0.064908	0.009305	0.022992

Tabla 5.4: Resultado de las métricas para la función ZDT3

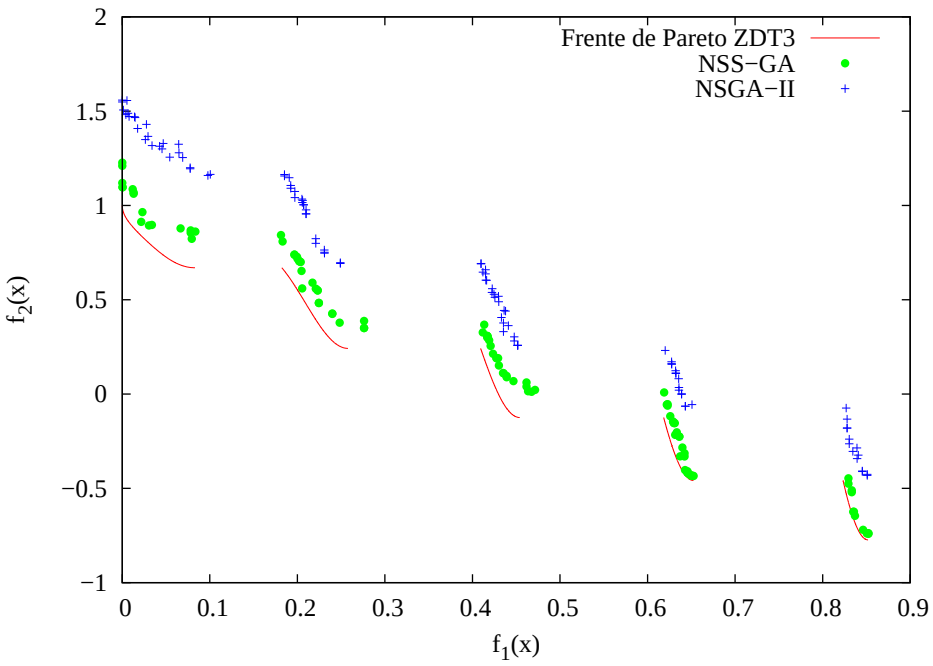

Figura 5.5: Comparación del NSS-GA y el NSGA-II en el problema multiobjetivo ZDT3

5.4.4. ZDT4

El problema multiobjetivo ZDT4, se caracteriza por tener 21^9 frentes de Pareto locales y con varios algoritmos evolutivos multiobjetivo, tienden a quedar atrapado en un frente de Pareto falso. Para este problema, también nuestro propuesta algorítmica es mejor en todas la métricas tal y como lo muestra la tabla 5.6. Aunque no todas las soluciones del NSS-GA dominan a las del NSGA-II, se tiene una mejor aproximación al verdadero frente, la cual se muestra gráficamente en la figura 5.6.

ZDT4				
Métrica	NSS-GA		NSGA-II	
	media	σ	*media*	σ
$\mathcal{DGI}$	0.122063	0.058813	0.156509	0.051699
$\mathcal{E}$	0.455495	0.416654	3.098866	2.822281
$\mathcal{C}$	0.686486	0.322281	0.332336	0.388879

Tabla 5.5: Resultado de las métricas para la función ZDT4

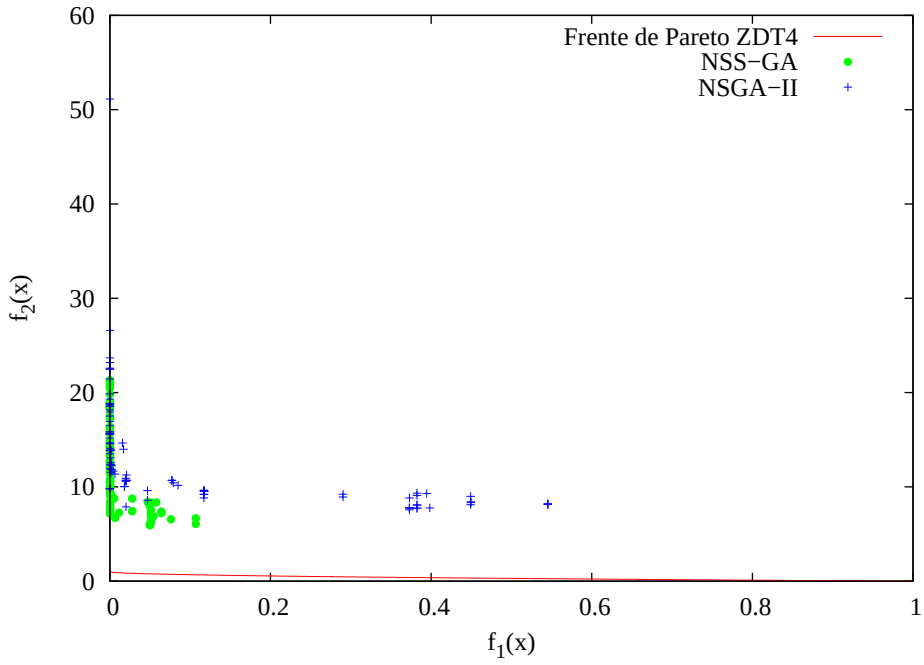

Figura 5.6: Comparación del NSS-GA y el NSGA-II en el problema multiobjetivo ZDT4

5.4.5. ZDT6

En la última función de prueba de los problemas ZDT, los resultados que se obtienen con el NSS-GA son favorables. La aproximación de las soluciones al verdadero frente de Pareto es mejor que las que se obtienen con el NSGA-II, tal y como lo muestra la figura 5.7. La tabla 5.6, muestra que nuestro algoritmo tiene una mejor aproximación ($\mathcal{DGI}$) al frente de Pareto verdadero, pero la dispersión de soluciones no es la más favorable ($\mathcal{E}$). Aunque la desviación estándar de los resultados obtenidos revelan que es más estable el NSGA-II, la diferencia es poco significativa en comparación con nuestro algoritmo, si bien para el caso de la métrica de espaciamiento, el NSGA-II le gana claramente a nuestro algoritmo.

ZDT6				
Métrica	NSS-GA		NSGA-II	
	media	σ	*media*	σ
$\mathcal{DGI}$	0.008980	0.004758	0.046699	0.007258
$\mathcal{E}$	0.171233	0.117406	0.106812	0.055624
$\mathcal{C}$	0.769754	0.273523	0.144094	0.230425

Tabla 5.6: Resultado de las métricas para la función ZDT6

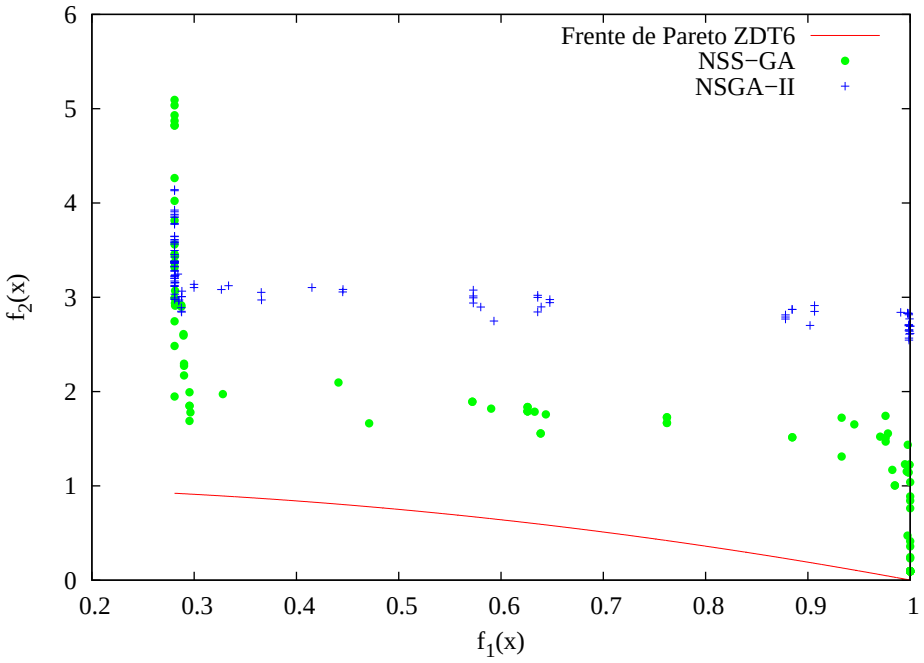

Figura 5.7: Comparación del NSS-GA y el NSGA-II en el problema multiobjetivo ZDT6

5.4.6. DTLZ1

La función DTLZ1 es la primera función de prueba de las 9 propuestas por Deb et al. en [17]. Este grupo de problemas multiobjetivo, se caracteriza por ser escalable en el sentido del número de funciones objetivo. Para las pruebas realizadas empleamos 3 funciones objetivo. Los resultados se muestran en la tabla 5.7. Los resultados favorecen a nuestra propuesta algorítmica en cuanto al acercamiento al verdadero frente de Pareto. La dispersión de las soluciones no es la más deseable, pero no hay que olvidar que nuestro objetivo principal es converger rápidamente hacia el verdadero frente. Sobre la cobertura de soluciones podemos concluir, que en la mayoría de las ejecuciones, el algoritmo NSS-GA logra dominar al NSGA-II. La figura 5.8 muestra que las soluciones del NSS-GA tiene un mejor acercamiento al verdadero frente de Pareto.

DTLZ1

Métrica	NSS-GA		NSGA-II	
	media	σ	*media*	σ
$\mathcal{DGI}$	0.658650	0.107311	0.779135	0.168162
$\mathcal{E}$	17.965977	7.564753	16.132116	6.874782
$\mathcal{C}$	0.590605	0.147409	0.222936	0.112005

Tabla 5.7: Resultado de las métricas para la función DTLZ1

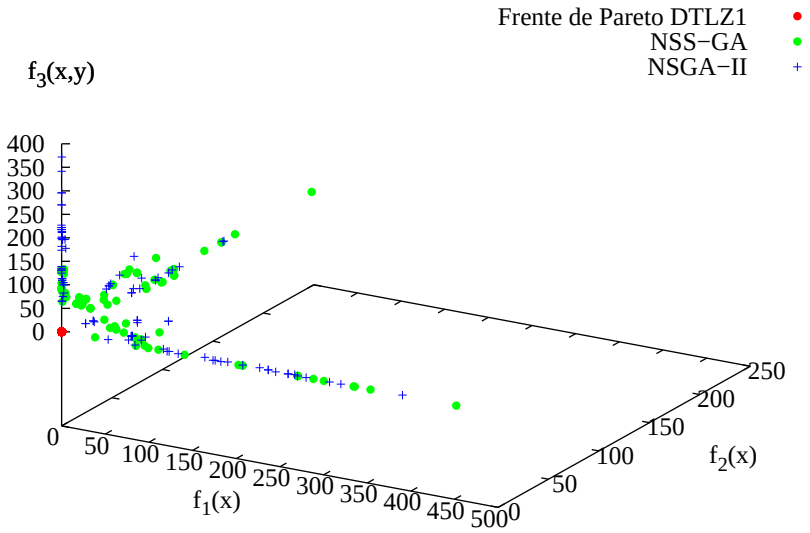

Figura 5.8: Comparación del NSS-GA y el NSGA-II en el problema multiobjetivo DTLZ1

5.4.7. DTLZ2

En la función DTLZ2 también de tres objetivos, la competencia entre los dos algoritmos resulta muy disputada. El algoritmo NSS-GA tiene por muy poco un mejor acercamiento al frente de Pareto verdadero. La tabla 5.8 revela que también para la dispersión de soluciones la competencia es apretada. El algoritmo NSGA-II es, por muy poco mejor que nuestra propuesta. En la cobertura de soluciones, podemos decir que nuestro algoritmo domina por muy poco las soluciones del NSGA-II. La figura 5.9 muestra la convergencia de los dos algoritmos hacia el verdadero frente de Pareto.

ZDT6

Métrica	NSS-GA		NSGA-II	
	media	σ	*media*	σ
$\mathcal{DGI}$	0.000403	0.000022	0.000428	0.000024
$\mathcal{E}$	0.055607	0.005740	0.055528	0.004754
$\mathcal{C}$	0.150000	0.072019	0.025667	0.022462

Tabla 5.8: Resultado de las métricas para la función DTLZ2

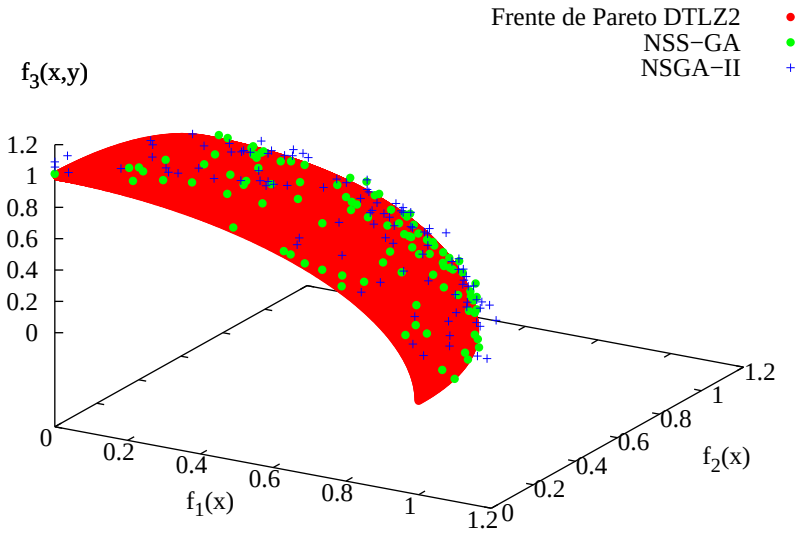

Figura 5.9: Comparación del NSS-GA y el NSGA-II en el problema multiobjetivo DTLZ2

5.5. Conclusiones sobre los resultados obtenidos

Nuestra propuesta NSS-GA resultó ser mejor que el algoritmo evolutivo multiobjetivo NSGA-II. El NSS-GA mostró ser un algoritmo que converge más rápido al frente de Pareto verdadero en todas las funciones de prueba adoptadas. Aunque la dispersión de las soluciones no es del todo satisfactoria, las diferencias no son realmente significativas en comparación con el NSGA-II.

En esta tesis, nuestro objetivo fundamental fue converger al frente de Pareto verdadero en un menor número de evaluaciones de la función objetivo. El algoritmo híbrido que hemos diseñado, logra en 4,000 evaluaciones ser mejor que el algoritmo multiobjetivo adoptado (el NSGA-II).

En términos generales, nuestra propuesta NSS-GA resultó ser un algoritmo consistente y muy competitivo sobre el conjunto de funciones adoptadas. La definición de los problemas multiobjetivo se presentan en el apéndice B. Mientras que las gráficas de convergencia de nuestro algoritmo híbrido hacia el frente de Pareto verdadero, se presentan en el apéndice C.

CAPÍTULO 6

CONCLUSIONES Y TRABAJO A FUTURO

6.1. Conclusiones

Con base en los resultados obtenidos en este trabajo, podemos concluir que hemos diseñado un nuevo algoritmo híbrido denominado *Nonlinear Simplex Search Genetic Algorithm* (NSS-GA) para optimización multiobjetivo. Este enfoque supera al algoritmo evolutivo base adoptado que fue el NSGA-II. Los resultados en todas métricas utilizadas, revelan que nuestra propuesta algorítmica es mejor cuando se usa un número reducido de evaluaciones de la función objetivo.

Nuestro trabajo principal, se centró en acoplar y hacer eficiente el método de Nelder-Mead para cada problema multiobjetivo abordado. Tomando en cuenta esto, concluimos que:

1. El éxito de convergencia del método de Nelder-Mead, depende directamente del simplex con que se inicie el proceso de optimización.
2. La construcción de un simplex regular, resulta ineficiente y, en la mayoría de los casos, inoperable para converger a un valor mejor, al menos en las funciones de prueba adoptadas.
3. La estrategia de reducir el espacio de búsqueda y la construcción del simplex inicial por medio de las secuencias de baja discrepancia, mejora considerablemente el desempeño del método de Nelder-Mead, en la búsqueda local.
4. El criterio de parada en la búsqueda local (ecuación 4.4), mejora el desempeño del algoritmo. En los experimentos realizados, la búsqueda local resulta poco eficiente, si se dejase iterar por un mayor número de veces.
5. En general, el diseño del mecanismo en la fase de explotación, agiliza la convergencia de las soluciones y trae consigo una reducción del número de evaluaciones de la función objetivo, para alcanzar el frente de Pareto verdadero.

6.2. Trabajo a futuro

Existen diversas estrategias e ideas, las cuales debido a las limitantes obvias de tiempo, no se llevaron a cabo, pero que probablemente ofrezcan mejoras al algoritmo. Algunas de ellas se comentan a continuación:

1. Introducir un mecanismo en el cual se pueda decidir cuándo ya no se deben realizar nuevas búsquedas locales.

2. Idear un nuevo método para reducir el dominio de la búsqueda, ya que la estrategia que proponemos puede resultar ineficiente cuando se pierde diversidad en la población. En otras palabras, si la muestra poblacional con la que se reduce el dominio de creación del simplex es similar, la estrategia de búsqueda puede resultar ineficiente.

3. Idear un método en el cual se reduzca la dimensión del simplex y optimizar con respecto a esta nueva dimensión, con la finalidad de converger más rápido a óptimos locales. De esta manera se explotará más rápidamente el espacio de búsqueda, trayendo como consecuencia converger más rápido a un óptimo global.

4. Utilizar las estrategias existentes de las modificaciones del método de Nelder-Mead como las descritas en [4, 58, 75], con la finalidad agilizar la convergencia hacia óptimos locales.

5. Adoptar otro algoritmo evolutivo multiobjetivo del estado del arte en la fase de exploración y ver los resultados que se obtienen con esta estrategia de búsqueda local.

6. Abordar el manejo de restricciones tanto para el caso mono-objetivo como para el multiobjetivo.

APÉNDICE A

FUNCIONES DE PRUEBA (MONO-OBJETIVO)

A.1. Modelo de esfera

$$f_1(\vec{x}) = \sum_{i=1}^{n} x_i^2 \tag{A.1}$$

donde $n = 30$ y $x_i \in [-100, 100]$. Tiene un mínimo mín$(f_1) = f_1(0, \ldots, 0) = 0$

A.2. Problema de Schwefel 2.22

$$f_2(\vec{x}) = \sum_{i=1}^{n} |x_i| + \prod_{i=1}^{n} |x_i| \tag{A.2}$$

donde $n = 30$ y $x_i \in [-10, 10]$. Tiene un mínimo mín$(f_2) = f_2(0, \ldots, 0) = 0$

A.3. Problema de Schwefel 1.2

$$f_3(\vec{x}) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j \right)^2 \tag{A.3}$$

donde $n = 30$ y $x_i \in [-100, 100]$. Tiene un mínimo mín$(f_3) = f_3(0, \ldots, 0) = 0$

A.4. Problema de Schwefel 2.21

$$f_4(\vec{x}) = \max_i\{|x_i|, 1 \leq i \leq n\} \tag{A.4}$$

donde $n = 30$ y $x_i \in [-100, 100]$. Tiene un mínimo mín$(f_4) = f_4(0, \ldots, 0) = 0$

A.5. Función generalizada de Rosenbrok

$$f_5(\vec{x}) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2] \tag{A.5}$$

donde $n = 30$ y $x_i \in [-30, 30]$. Tiene un mínimo mín$(f_5) = f_5(1, \ldots, 1) = 0$

A.6. Función escalón

$$f_6(\vec{x}) = \sum_{i=1}^{n} (\lfloor x_i + 0.5 \rfloor)^2 \tag{A.6}$$

donde $n = 30$ y $x_i \in [-100, 100]$. Tiene un mínimo mín$(f_6) = f_6(0, \ldots, 0) = 0$

A.7. Función Quartic

$$f_7(\vec{x}) = \sum_{i=1}^{n} i x_i^4 + random[0, 1) \tag{A.7}$$

donde $n = 30$ y $x_i \in [-1.28, 1.28]$. Tiene un mínimo mín$(f_7) = f_7(0, \ldots, 0) = 0$

A.8. Problema generalizado de Schwefel 2.26

$$f_8(\vec{x}) = -\sum_{i=1}^{n} (x_i \sin(\sqrt{|x_i|})) \tag{A.8}$$

donde $n = 30$ y $x_i \in [-500, 500]$. Tiene un mínimo mín$(f_8) = f_8(420.9687, \ldots, 420.9687) = -12569.5$

A.9. Función generalizada de Rastrigin

$$f_9(\vec{x}) = \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i) + 10] \tag{A.9}$$

donde $n = 30$ y $x_i \in [-5.12, 5.12]$. Tiene un mínimo mín$(f_9) = f_9(0,\ldots,0) = 0$

A.10. Función de Ackley

$$f_{10}(\vec{x}) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos 2\pi x_i\right) + 20 + e \tag{A.10}$$

donde $n = 30$ y $x_i \in [-32, 32]$. Tiene un mínimo mín$(f_{10}) = f_{10}(0,\ldots,0) = 0$

A.11. Función generalizada de Griewank

$$f_{11}(\vec{x}) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \tag{A.11}$$

donde $n = 30$ y $x_i \in [-600, 600]$. Tiene un mínimo mín$(f_{11}) = f_{11}(0,\ldots,0) = 0$

A.12. Funciones generalizadas de penalización

A.12.1. Función generalizada de penalización 1.1

$$\begin{aligned} f_{12}(\vec{x}) &= \frac{\pi}{n}\left\{10\sin^2(\pi y_i) + \sum_{i=1}^{n-1}(y_i - 1)^2[1 + 10\sin^2(\pi y_{i+1})] + (y_n - 1)^2\right\} \\ &\quad + \sum_{i=1}^{n} u(x_i, 10, 100, 4) \end{aligned} \tag{A.12}$$

para $n = 30$ y $x_i \in [-50, 50]$. Tiene un mínimo mín$(f_{12}) = f_{12}(1,\ldots,1) = 0$

A.12.2. Función generalizada de penalización 1.2

$$\begin{aligned} f_{13}(\vec{x}) &= 0.1\left\{\sin^2(3\pi x_1) + \sum_{i=1}^{n-1}(x_i - 1)^2[1 + \sin^2(3\pi x_{i+1})] + (y_n - 1)^2\left[1 + \sin^2(2\pi x_n)\right]\right\} \\ &\quad + \sum_{i=1}^{n} u(x_i, 5, 100, 4) \end{aligned} \tag{A.13}$$

para $n = 30$ y $x_i \in [-50, 50]$. Tiene un mínimo mín$(f_{13}) = f_{13}(1, \ldots, 1) = 0$ donde:

$$\begin{aligned} y_i &= 1 + \tfrac{1}{4}(x_i + 1) \\ u(x_i, a, k, m) &= \begin{cases} k(x_i - a)^m, & x_i > a, \\ 0, & -a \leq x_i \leq a, \\ k(-x_i - a)^m, & x_i < -a \end{cases} \end{aligned}$$

A.13. Función trinchera de Shekel

$$f_{14}(\vec{x}) = \left[\frac{1}{500} + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2} (x_i - a_{ij})^6} \right]^{-1} \tag{A.14}$$

donde $x_i \in [-65.536, 65.536]$. Tiene un mínimo mín$(f_{14}) = f_{14}(-32, -32) \approx 1$.

A.14. Función de Kowalik

$$f_{15}(\vec{x}) = \sum_{i=1}^{11} \left[a_i - \frac{x_1(b_i^2 + b_i x_2)}{b_i^2 + b_i x_3 + x_4} \right]^2. \tag{A.15}$$

donde $x_i \in [-5, 5]$. Tiene un mínimo mín$(f_{15}) \approx f_{15}(0.1928, 0.1908, 0.1231, 0.1358) \approx 0.0003075$.

A.15. Función Six-Hump Camel-Back

$$f_{16}(\vec{x}) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1 x_2 - 4x_2^2 + 4x_2^4 \tag{A.16}$$

donde $x_i \in [-5, 5]$. Tiene un mínimo mín$(f_{15}) = f_{16}(0.08983, -0.7126) = f_{16}(-0.08983, 0.7126) = -1.0316285$.

A.16. Función de Branin

$$f_{17}(\vec{x}) = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6 \right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos x_1 + 10 \tag{A.17}$$

donde $x_1 \in [-5, 10]$ y $x_2 \in [0, 15]$. Con un mínimo en $\vec{x}_{\text{mín}} = \{(3.142, 2.275), (3.142, 2.275), (9,425, 2.425)\}$, $f(\vec{x}_{\text{mín}}) = 0.398$

A.17. Función de Goldstein-Price

$$\begin{aligned} f_{18}(\vec{x}) &= \left[1+(x_1+x_2+1)^2(19-14x_1+3x_1^2-14x_2+6x_1x_2+3x_2^2)\right] \\ &\quad \times \left[30+(2x_1-3x_2)^2(18-32x_1+12x_1^2+48x_2-36x_1x_2+27x_2^2)\right] \end{aligned} \tag{A.18}$$

donde $x_i \in [-2,2]$. Tiene un mínimo mín$(f_{18}) = f18(0,-1) = 3$.

A.18. Familia de Hartman

A.18.1. Familia de Hartman 1.1

$$f_{19}(\vec{x}) = -\sum_{i=1}^{4} c_i \exp\left[-\sum_{j=1}^{n} a_{ij}(x_j - p_{ij})^2\right] \tag{A.19}$$

donde $n = 3$ y $x_i \in [0,1]$. Tiene un mínimo mín$(f_{19}) = f_{19}(0.114, 0.556, 0.852) = -3.86$.

A.18.2. Familia de Hartman 1.2

$$f_{20}(\vec{x}) = -\sum_{i=1}^{n} c_i \exp\left[-\sum_{j=1}^{n} a_{ij}(x_j - p_{ij})^2\right] \tag{A.20}$$

donde $n = 6$ y $x_i \in [0,10]$. Tiene un mínimo mín$(f_{20}) = f_{20}(0.201, 0.150, 0.477, 0.275, 0.311, 0.657) = -3.32$.

A.19. Familia de Shekel

A.19.1. Familia de Shekel 1.1

$$f_{21}(\vec{x}) = -\sum_{i=1}^{n} [(\vec{x}-a_i)(\vec{x}-a_i)^T + c_i]^{-1} \tag{A.21}$$

donde $n = 5$ y $x_i \in [0,10]$. Con mínimo en $\vec{x}_{\text{mín}} \approx a_i$, $f_{21}(\vec{x}_{\text{mín}}) \approx \frac{1}{c_i}$ para $1 \leq i \leq n$.

A.19.2. Familia de Shekel 1.2

$$f_{22}(\vec{x}) = -\sum_{i=1}^{n} [(\vec{x}-a_i)(\vec{x}-a_i)^T + c_i]^{-1} \tag{A.22}$$

donde $n = 7$ y $x_i \in [0,10]$. Con mínimo en $\vec{x}_{\text{mín}} \approx a_i$, $f_{22}(\vec{x}_{\text{mín}}) \approx \frac{1}{c_i}$ para $1 \leq i \leq n$.

i	a_i	b_i^{-1}
1	0.1957	0.25
2	0.1947	0.5
3	0.1735	1
4	0.1600	2
5	0.0844	4
6	0.0627	6
7	0.0456	8
8	0.0342	10
9	0.0323	12
10	0.0235	14
11	0.0246	16

Tabla A.1: Función de Kowalk f_{15}

i	$a_{ij}, j = 1,2,3$			c_i	$p_{ij}, j = 1,2,3$		
1	3	10	30	1	0.3689	0.1170	0.2673
2	0.1	10	35	1.2	0.4699	0.4387	0.7470
3	3	10	30	3	0.1091	0.8732	0.5547
4	0.1	10	35	3.2	0.038150	0.5743	0.8828

Tabla A.2: Función de Hartman f_{19}

A.19.3. Familia de Shekel 1.3

$$f_{23}(\vec{x}) = -\sum_{i=1}^{n}[(\vec{x}-a_i)(\vec{x}-a_i)^T + c_i]^{-1} \tag{A.23}$$

donde $n = 10$ y $x_i \in [0,10]$. Con mínimo en $\vec{x}_{\text{mín}} \approx a_i$, $f_{23}(\vec{x}_{\text{mín}}) \approx \frac{1}{c_i}$ para $1 \leq i \leq n$.

i	$a_{ij}, j = 1,\ldots,6$						c_i	$p_{ij}, j = 1,\ldots,6$					
1	10	3	17	3.5	1.7	8	1	0.1312	0.1696	0.5569	0.0124	0.8283	0.5886
2	0.05	10	17	0.1	8	14	1.2	0.2329	0.4135	0.8307	0.3736	0.1004	0.9991
3	3	3.5	1.7	10	17	8	3	0.2348	0.1415	0.3522	0.2883	0.3047	0.6650
4	17	8	0.05	10	0.1	14	3.2	0.4047	0.8828	0.8732	0.5743	0.1091	0.0381

Tabla A.3: Función de Hartman f_{20}

i	$a_{ij}, j = 1, \ldots, 4$				c_i
1	4	4	4	4	0.1
2	1	1	1	1	0.2
3	8	8	8	8	0.2
4	6	6	6	6	0.4
5	3	7	3	7	0.4
6	2	9	2	9	0.6
7	5	5	3	3	0.3
8	8	1	8	1	0.7
9	6	2	6	2	0.5
10	7	3.6	7	3.6	0.5

Tabla A.4: Funciones de Shekel f_{21}, f_{22} y f_{23}

APÉNDICE B

FUNCIONES DE PRUEBA (MULTI-OBJETIVO)

B.1. Función ZDT1

Este problema multi-objetivo fue propuesto por Zitzler [85] y consiste en minimizar:

$$\vec{f}(\vec{x}) = [f_1(\vec{x}), f_2(\vec{x})] \tag{B.1}$$

donde:

$$\begin{aligned} f_1(\vec{x}) &= x_1 \\ f_2(\vec{x}) &= g(\vec{x}) \cdot h(f_1(\vec{x}), g(\vec{x})) \\ g(\vec{x}) &= 1 + \frac{9}{(n-1)} \sum_{i=2}^{n} x_i \\ h(f_1(\vec{x}), g(\vec{x})) &= 1 - \sqrt{\frac{f_1(\vec{x})}{g(\vec{x})}} \end{aligned}$$

para $n = 30$ y $x_i \in [0,1]$. Tiene un frente de Pareto convexo y conectado.

B.2. Función ZDT2

Este problema multi-objetivo fue propuesto por Zitzler [85] y consiste en minimizar:

$$\vec{f}(\vec{x}) = [f_1(\vec{x}), f_2(\vec{x})] \tag{B.2}$$

donde:

$$\begin{aligned} f_1(\vec{x}) &= x_1 \\ f_2(\vec{x}) &= g(\vec{x}) \cdot h(f_1(\vec{x}), g(\vec{x})) \\ g(\vec{x}) &= 1 + \frac{9}{(n-1)} \sum_{i=2}^{n} x_i \\ h(f_1(\vec{x}), g(\vec{x})) &= 1 - \left(\frac{f_1(\vec{x})}{g(\vec{x})} \right)^2 \end{aligned}$$

para $n = 30$ y $x_i \in [0,1]$. Tiene un frente de Pareto no convexo y conectado.

B.3. Función ZDT3

Este problema multi-objetivo fue propuesto por Zitzler [85] y consiste en minimizar:

$$\vec{f}(\vec{x}) = [f_1(\vec{x}), f_2(\vec{x})] \tag{B.3}$$

donde:

$$\begin{aligned} f_1(\vec{x}) &= x_1 \\ f_2(\vec{x}) &= g(\vec{x}) \cdot h(f_1(\vec{x}), g(\vec{x})) \\ g(\vec{x}) &= 1 + \frac{9}{(n-1)} \sum_{i=2}^{n} x_i \\ h(f_1(\vec{x}), g(\vec{x})) &= 1 - \sqrt{\frac{f_1(\vec{x})}{g(\vec{x})}} - \left(\frac{f_1(\vec{x})}{g(\vec{x})} \right) \sin(10\pi f_1(\vec{x})) \end{aligned}$$

para $n = 30$ y $x_i \in [0,1]$. Tiene un frente de Pareto discontinuo.

B.4. Función ZDT4

Este problema multi-objetivo fue propuesto por Zitzler [85] y consiste en minimizar:

$$\vec{f}(\vec{x}) = [f_1(\vec{x}), f_2(\vec{x})] \tag{B.4}$$

donde:

$$\begin{aligned} f_1(\vec{x}) &= x_1 \\ f_2(\vec{x}) &= g(\vec{x}) \cdot h(f_1(\vec{x}), g(\vec{x})) \\ g(\vec{x}) &= 1 + 10(n-1) + \sum_{i=2}^{n} (x_i^2 - 10\cos(4\pi x_i)) \\ h(f_1(\vec{x}), g(\vec{x})) &= 1 - \sqrt{\frac{f_1(\vec{x})}{g(\vec{x})}} \end{aligned}$$

para $n = 10$, $x_1 \in [0,1]$ y $x_2, \ldots, x_n \in [-5,5]$. Tiene un frente de Pareto convexo, conectado y existen 21^9 frentes de Pareto locales.

B.5. Función ZDT6

Este problema multi-objetivo fue propuesto por Zitzler [85] y consiste en minimizar:

$$\vec{f}(\vec{x}) = [f_1(\vec{x}), f_2(\vec{x})] \tag{B.5}$$

donde:

$$\begin{aligned} f_1(\vec{x}) &= 1 - \exp(-4x_1) \cdot \sin^6(6\pi x_1) \\ f_2(\vec{x}) &= g(\vec{x}) \cdot h(f_1(\vec{x}), g(\vec{x})) \\ g(\vec{x}) &= 1 + 9 \left(\frac{\sum_{i=2}^{n} x_i}{(n-1)} \right)^{0.25} \\ h(f_1(\vec{x}), g(\vec{x})) &= 1 - \left(\frac{f_1(\vec{x})}{g(\vec{x})} \right)^2 \end{aligned}$$

para $n = 10$ y $x_i \in [0,1]$. Tiene un frente de Pareto convexo y continuo.

B.6. Función DTLZ1

Este problema multi-objetivo fue propuesto por Deb [17] y consiste en minimizar:

$$\vec{f}(\vec{x}) = [f_1(\vec{x}), f_2(\vec{x}), f_2(\vec{x})] \tag{B.6}$$

donde:

$$\begin{aligned} f_1(\vec{x}) &= \frac{1}{2} x_1 x_2 [1 + g(\vec{x})] \\ f_2(\vec{x}) &= \frac{1}{2} x_1 (1 - x_2) [1 + g(\vec{x})] \\ f_3(\vec{x}) &= \frac{1}{2} (1 - x_1) [1 + g(\vec{x})] \\ g(\vec{x}) &= 100 \left\{ n + \sum_{i=3}^{n} (x_i - 0.5)^2 - \cos[20\pi(x_i - 0.5)] \right\} \end{aligned}$$

para $n = 12$ y $x_i \in [0,1]$. Tiene un frente de Pareto lineal y contiene $11^n - 1$ frentes de Pareto locales.

B.7. Función DTLZ2

Este problema multi-objetivo fue propuesto por Deb [17] y consiste en minimizar:

$$\vec{f}(\vec{x}) = [f_1(\vec{x}), f_2(\vec{x}), f_2(\vec{x})] \tag{B.7}$$

donde:

$$\begin{aligned} f_1(\vec{x}) &= \cos\left(\frac{\pi}{2}x_1\right)\cos\left(\frac{\pi}{2}x_2\right)[1+g(\vec{x})] \\ f_2(\vec{x}) &= \cos\left(\frac{\pi}{2}x_1\right)\sin\left(\frac{\pi}{2}x_2\right)[1+g(\vec{x})] \\ f_3(\vec{x}) &= \sin\left(\frac{\pi}{2}x_1\right)[1+g(\vec{x})] \\ g(\vec{x}) &= \sum_{i=3}^{n}(x_i-0.5)^2 \end{aligned}$$

para $n = 12$ y $x_i \in [0,1]$. Tiene un frente de Pareto convexo y conectado.

APÉNDICE C

CONVERGENCIA EN LOS PROBLEMAS MULTI-OBJETIVO

En este apéndice presentamos las gráficas de nuestro algoritmo NSS-GA, con la finalidad de validar la convergencia al frente de Pareto verdadero en un determinado número de evaluaciones de la función objetivo.

Para las funciones ZDT1, ZDT2, ZDT3, ZDT6 y DTLZ2 con tan solo 7,000 evaluaciones de la función objetivo, nuestra propuesta converge al frente de Pareto verdadero. Mientras que se necesitaron 16,000 para converger en la función ZDT4 y 70,000 para la DTLZ1.

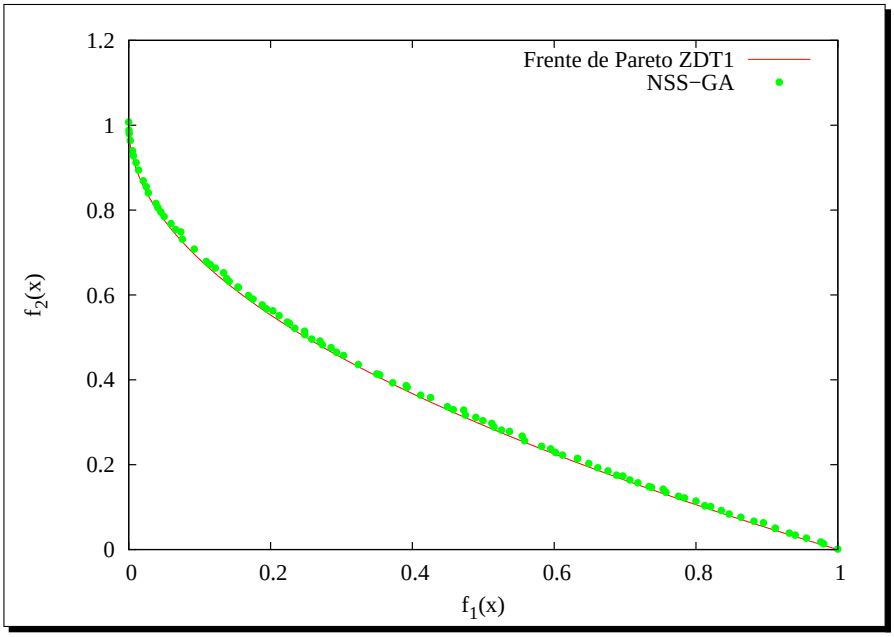

Figura C.1: Convergencia al frente de Pareto verdadero en la función ZDT1 con 7,000 evaluaciones de la función objetivo

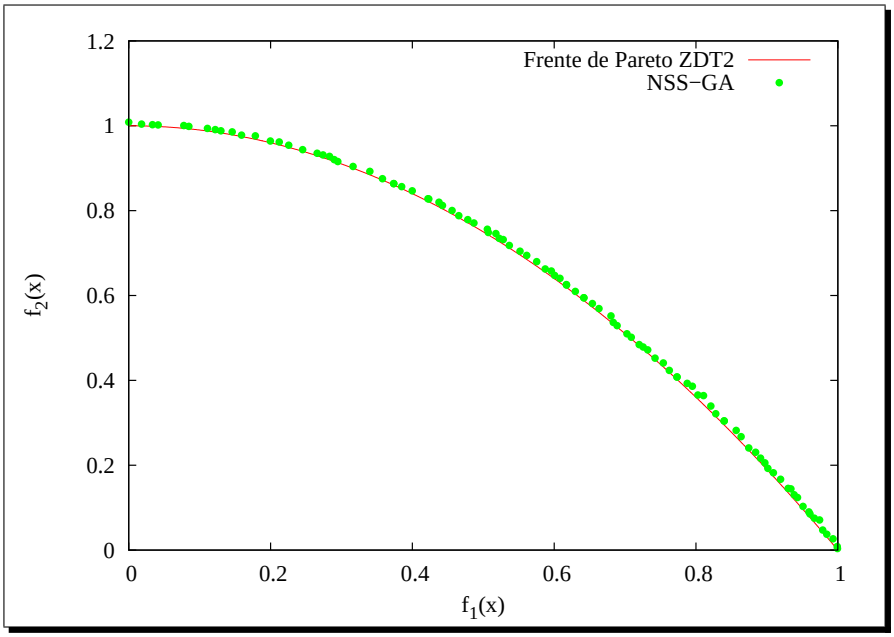

Figura C.2: Convergencia al frente de Pareto verdadero en la función ZDT2 con 7,000 evaluaciones de la función objetivo

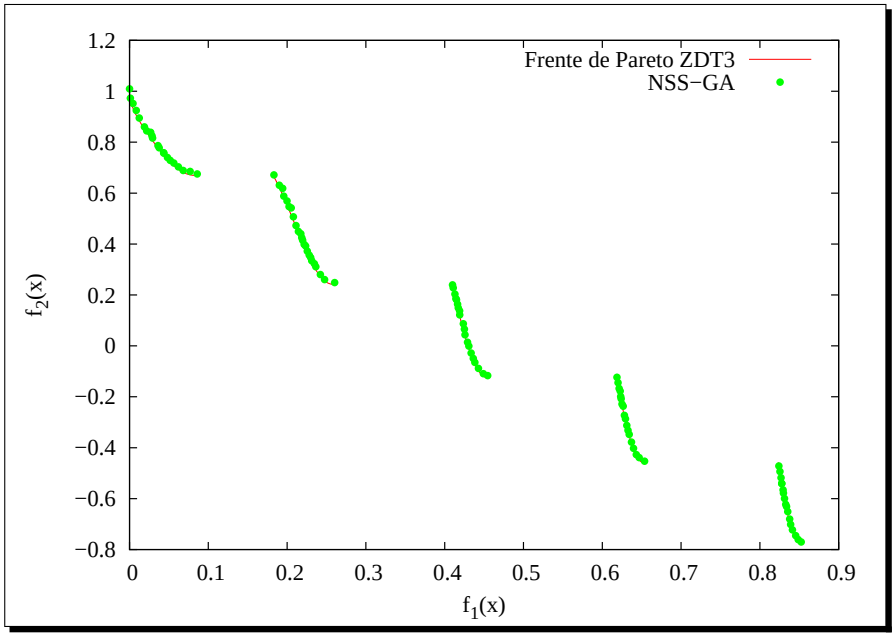

Figura C.3: Convergencia al frente de Pareto verdadero en la función ZDT3 con 7,000 evaluaciones de la función objetivo

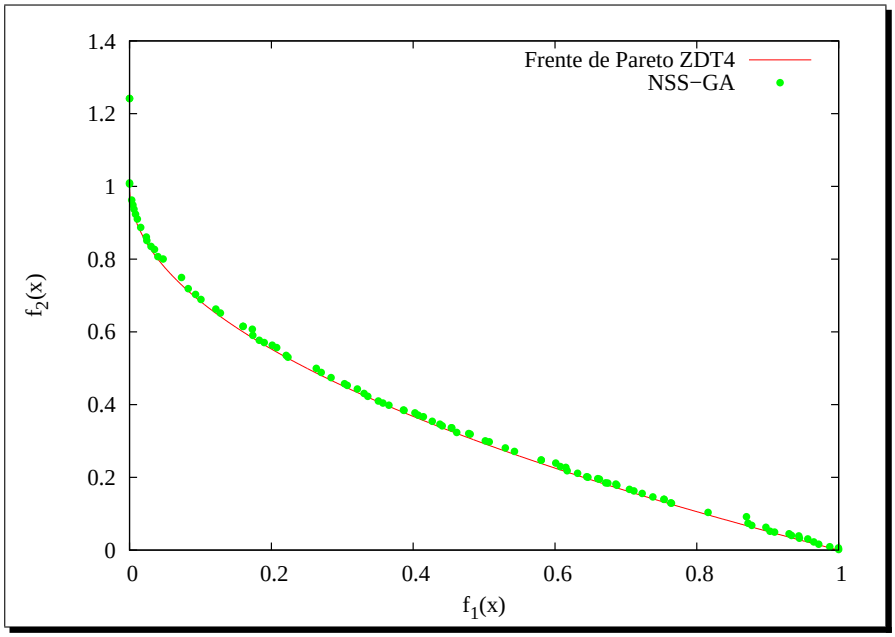

Figura C.4: Convergencia al frente de Pareto verdadero en la función ZDT4 con 16,000 evaluaciones de la función objetivo

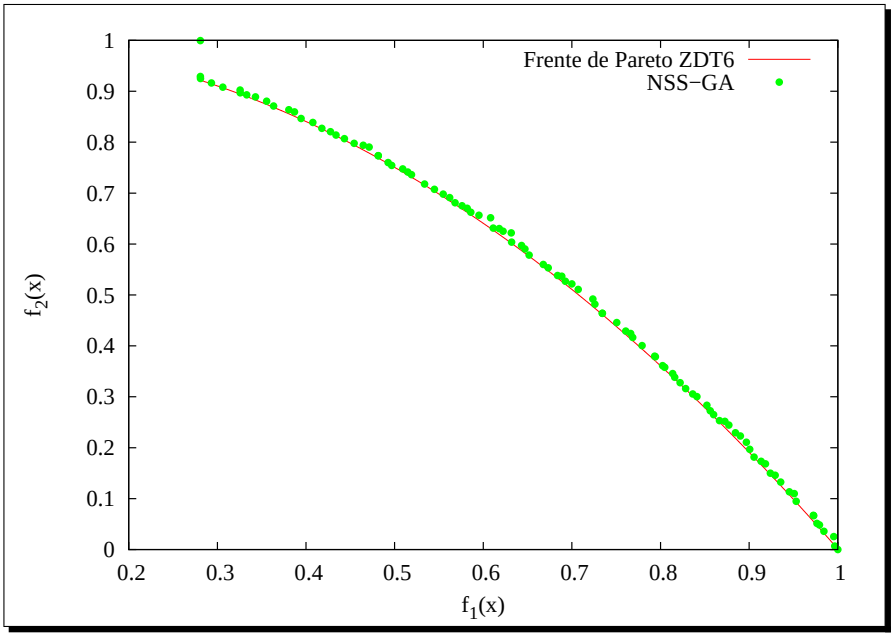

Figura C.5: Convergencia al frente de Pareto verdadero en la función ZDT6 con 7,000 evaluaciones de la función objetivo

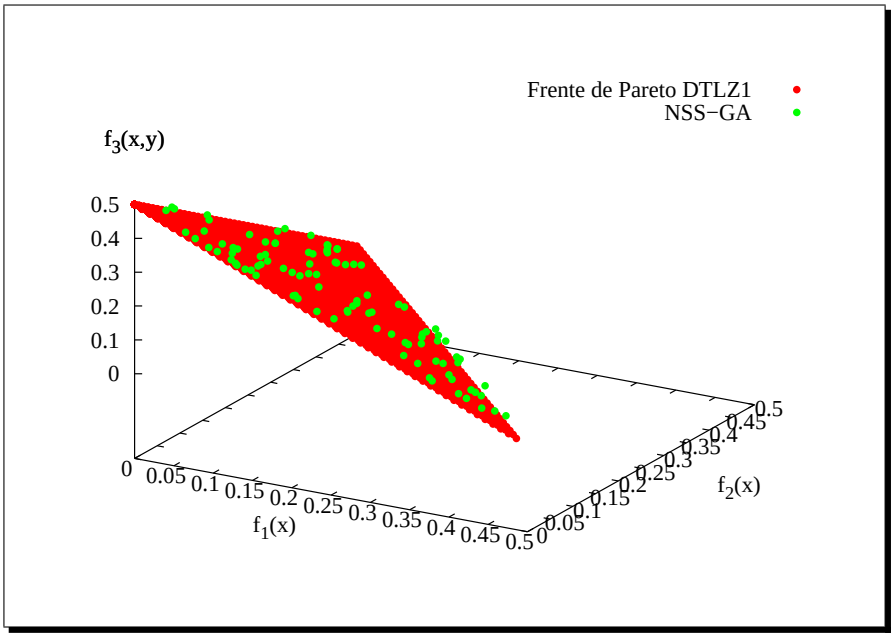

Figura C.6: Convergencia al frente de Pareto verdadero en la función DTLZ1 con 70,000 evaluaciones de la función objetivo

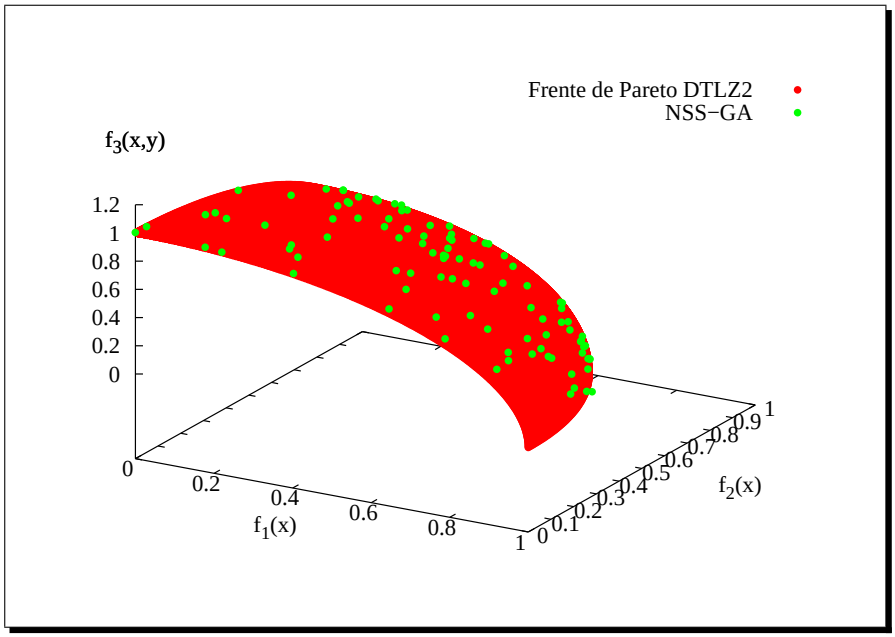

Figura C.7: Convergencia al frente de Pareto verdadero en la función DTLZ2 con 7,000 evaluaciones de la función objetivo

BIBLIOGRAFÍA

[1] Thomas Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, Oxford, UK, 1996.

[2] James Mark Baldwin. A New Factor in Evolution. *The American Naturalist*, 30(354):441–451, 1896.

[3] Vincent Barichard and Jin-Kao Hao. Genetic Tabu Search for the Multi-objective Knapsack Problem. *Journal of Tsinghua Siece and Technology*, 8(1):8–13, 2003.

[4] Russell R. Barton and Jr. John S. Ivey. Nelder-mead simplex modifications for simulation optimization. *Manage. Sci.*, 42(7):954–973, 1996.

[5] George E. P. Box. Evolutionary Operation: A Method for Increasing Industrial Productivity. *Journal of Applied Statistics*, 6(2):81–101, June 1957.

[6] Walter D. Cannon. *The Wisdom of the body*. Norton and Company, New York, 1932.

[7] A. Carlisle and G. Dozier. An off-the-self PSO. In *Proceedings of Workshop on Particle Swarm Optimization*, Indianapolis, 2001.

[8] Abraham Charnes and William W. Cooper. *Management models and industrial applications of linear programming*. John Wiley, New York, 1961.

[9] Rachid Chelouah and Patrick Siarry. Genetic and Nelder-Mead algorithms hybridized for a more accurate global optimization of continuous multiminima functions. *European Journal of Operational Research*, 148(2):335–348, July 2003.

[10] Carlos A. Coello Coello, Gary B. Lamont, and David A. Van Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Springer, 2nd edition, 2007.

[11] J. L. Cohon and D. H. Marks. review and Evaluation of Multiobjective Programming Techniques. *Water Resources Research*, 11(2):208–220, 1975.

[12] Jared L. Cohon. *Multiobjective Programming and Planning*. Academic Press, 1978.

[13] Lawrence Davis. Adapting operator probabilities in genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 61–69, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[14] Kalyanmoy Deb. *Optimization for Engineering Design: Algorithms and Examples*. Prentice-Hall of India Pvt. Ltd., 2002.

[15] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions Evolutionary Computation*, 6(2):182–197, 2002.

[16] Kalyanmoy Deb and David E. Goldberg. An Investigation of Niche and Species Formation in Genetic Function Optimization. pages 42–50, 1989.

[17] Kalyanmoy Deb, Lothar Thiele, Marco Laumanns, and Eckart Zitzler. Scalable Test Problems for Evolutionary Multi-Objective Optimization. Technical Report 112, Zurich, Switzerland, 2001.

[18] Marco Dorigo and Gianni Di Caro. The ant colony optimization meta-heuristic. In *New ideas in optimization*, pages 11–32. McGraw-Hill Ltd., UK, Maidenhead, UK, England, 1999.

[19] Lucien Duckstein. Multiobjective Optimization in Structural Design: The Model Choice Problem. In *Atrek, E., Gallagher, R. H., Ragsdell, K. M., and Zienkiewicz, O. C., editors. New Directions in Optimum Structural Design*, pages 459–481. John Wiley & Sons, Inc., 1984.

[20] Francis Ysidro Edgeworth. *Mathematical Psychics: An Essay on the Application of Mathematics to the Moral Sciences*. C. Kegan Paul and Co., London, 1881.

[21] Agoston E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. SpringerVerlag, 2003.

[22] L. J. Fogel. *Intelligence through simulated evolution: forty years of evolutionary programming*. John Wiley & Sons, Inc., New York, NY, USA, 1999.

[23] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution*. John Wiley, New York, USA, 1966.

[24] Carlos M. Fonseca and Peter J. Fleming. Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 416–423, San Mateo, California, 1993. University of Illinois at Urbana-Champaign, Morgan Kauffman Publishers.

[25] Fred Glover and Manuel Laguna. *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

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

[27] J. H. Halton. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerische Mathematik*, 2:84–90, december 1960.

[28] J. M. Hammersley. Monte-Carlo methods for solving multivariable problems. *Annals of the New York Academy of Science*, 86:844–874, 1960.

[29] Geoffrey E. Hinton and Steven J. Nowlan. How learning can guide evolution. In *Adaptive individuals in evolving populations: models and algorithms*, pages 447–454. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1996.

[30] John H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, Michigan, 1975.

[31] Jeffrey Horn, Nicholas Nafpliotis, and David E. Goldberg. A Niched Pareto Genetic Algorithm for Multiobjective Optimization. In *Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, volume 1, pages 82–87, Piscataway, New Jersey, 1994. IEEE Service Center.

[32] Xiaolin Hu, Zhangcan Huang, and Zhongfan Wang. Hybridization of the Multi-Objective Evolutionary Algorithms and the Gradient-based Algorithms. In *Proceedings of the 2003 Congress on Evolutionary Computation (CEC'2003)*, volume 2, pages 870–877, Canberra, Australia, December 2003. IEEE Press.

[33] Hisao Ishibuchi and Tadahiko Murata. Multi-Objective Genetic Local Search Algorithm and Its Application to Flowshop Scheduling. *IEEE Transactions on Systems, Man and Cybernetics*, 28(3):392–403, August 1998.

[34] J. Kennedy and R. Eberhart. Particle swarm optimization. In *IEEE International Conference on Neural Network*, volume 4, pages 1942–1948, 1995.

[35] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 1983.

[36] Joshua D. Knowles. ParEGO: a hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Transactions Evolutionary Computation*, 10(1):50–66, 2006.

[37] Joshua D. Knowles and David W. Corne. The Pareto Archived Evolution Strategy: A New Baseline Algorithm for Multiobjective Optimisation. In *1999 Congress on Evolutionary Computation*, pages 98–105, Washington, D.C., July 1999. IEEE Service Center.

[38] Joshua D. Knowles and David W. Corne. Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy. *Evolutionary Computation*, 8(2):149–172, 2000.

[39] Joshua D. Knowles and David W. Corne. M-PAES: A Memetic Algorithm for Multiobjective Optimization. In *2000 Congress on Evolutionary Computation*, volume 1, pages 325–332, Piscataway, New Jersey, July 2000. IEEE Service Center.

[40] Praveen Koduru, Sanjoy Das, and Stephen Welch. A Particle Swarm Optimization-Nelder Mead Hybrid Algorithm for Balanced Exploration and Exploitation in Multidimensional Search Space. In Hamid R. Arabnia, editor, *IC-AI*, pages 457–464. CSREA Press, 2006.

[41] Praveen Koduru, Sanjoy Das, and Stephen Welch. Multi-objective hybrid PSO using ε-fuzzy dominance. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and Evolutionary Computation*, pages 853–860, New York, NY, USA, 2007. ACM Press.

[42] Praveen Koduru, Sanjoy Das, Stephen Welch, and Judith L. Roe. Fuzzy Dominance Based Multi-objective GA-Simplex Hybrid Algorithms Applied to Gene Network Models. In Kalyanmoy Deb et al., editor, *Genetic and Evolutionary Computation–GECCO 2004. Proceedings of the Genetic and Evolutionary Computation Conference. Part I*, pages 356–367, Seattle, Washington, USA, June 2004. Springer-Verlag, Lecture Notes in Computer Science Vol. 3102.

[43] J. C. Lagarias, J. A. Reeds, M. H. Wright, and P. E. Wright. Convergence properties of the Nelder–Mead simplex method in low dimensions. *SIAM Journal of Optimization*, 9:112–147, 1998.

[44] Marco A. Luersen. and Rodolphe Le Riche. Globalized Nelder-Mead method for engineering optimization. *Computers & Structures*, 82:2251–2260, 2004.

[45] Changtong Luo and Bo Yu. Low Dimensional Simplex Evolution–A Hybrid Heuristic for Global Optimization. In *Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*, volume 2, pages 470–474, 2007.

[46] J. B. MacQueen. Some Methods for Classification and Analysis of Multivariate Observations. In *Proceedings of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.

[47] R. Marti and J. Marcos Moreno-Vega. Métodos multiarranque. *Inteligencia Artificial, Revista Iberoamericana de IA*, 7(19):49–60, 2003.

[48] K. I. M. McKinnon. Convergence of the Nelder–Mead Simplex Method to a Nonstationary Point. *SIAM Journal on Optimization*, 9(1):148–158, 1998.

[49] Kaisa M. Miettinen. *Nonlinear Multiobjective Optimization*. Kluwer Academic Publishers, Boston, Massachusetts, EE. UU., 1998.

[50] JoelÑ. Morse. Reducing the size of the nondominated set: Pruning by clustering. *Computers & Operations Research*, 7(1-2):55–66, 1980.

[51] Pablo Moscato. On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms. Technical Report Caltech Concurrent Computation Program, Report. 826, California Institute of Technology, Pasadena, California, USA, 1989.

[52] Pablo Moscato. Memetic algorithms: a short introduction. In *New ideas in optimization*, pages 219–234. McGraw-Hill Ltd., UK, Maidenhead, UK, England, 1999.

[53] J. A. Nelder and R. Mead. A Simplex Method for Function Minimization. *The Computer Journal*, 7:308–313, 1965.

[54] Andrzej Osyczka. *Multicriterion optimization in engineering with FORTRAN programs*. Ellis Horwood series in engineering science. 1984.

[55] Vilfredo Pareto. *Cours d' Economie Politique*, volume I and II. F. Rouge, Lausanne, 1896.

[56] Kenneth V. Price. An introduction to differential evolution. In *New ideas in optimization*, pages 79–108. McGraw-Hill Ltd., UK, Maidenhead, UK, England, 1999.

[57] Kenneth V. Price, Rainer M. Storn, and Jouni A. Lampinen. *Differential Evolution: A Practical Approach to Global Optimization*. Springer, 2005.

[58] M. K. Rahman. An intelligent moving object optimization algorithm for design problems with mixed variables, mixed constraints and multiple objectives. *Structural and Multidisciplinary Optimization*, 32(1):40–58, July 2006.

[59] Singiresu S. Rao. *Engineering Optimization*. John Wiley & Sons, Inc., 3rd edition, 1996.

[60] Real Academia Española. *Diccionario de la Lengua Española*. Espasa Calpe Mexicana, S.A., 22nd edition, 2001.

[61] Ingo Rechenberg. *Cybernetic Solution Path of an Experimental Problem*. Royal Aircraft Establishment, August 1965.

[62] R. S. Rosenberg. *Simulation of genetic populations with biochemical properties*. PhD thesis, University of Michigan, Ann Harbor, Michigan, 1967.

[63] Günter Rudolph. Convergence analysis of canonical genetic algorithms. *IEEE Transactions on Neural Networks*, 5(1):96–101, January 1994.

[64] J. David Schaffer. *Multiple Objective Optimization with Vector Evaluated Genetic Algorithms*. PhD thesis, Vanderbilt University, 1984.

[65] J. David Schaffer and Amy Morishima. An adaptive crossover distribution mechanism for genetic algorithms. In *Proceedings of the Second International Conference on Genetic Algorithms and their application*, pages 36–40, Mahwah, NJ, USA, 1987. Lawrence Erlbaum Associates, Inc.

[66] Jason R. Schott. Fault Tolerant Design Using Single and Multicriteria Genetic Algorithm Optimization. Master's thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, Massachusetts, May 1995.

[67] Hans-Paul Schwefel. *Kybernetische Evolution als Strategie der Experimentellen Forschung in der Stromungstechnik*. PhD thesis, Technical University of Berlin, 1965.

[68] Hans-Paul Schwefel. *Numerical Optimization of Computer Models*. John Wiley & Sons, Inc., New York, NY, USA, 1981.

[69] W. Spendley, G. R. Hext, and F. R. Himsworth. Sequential Application of Simplex Designs in Optimization and Evolutionary Operation. *Technometrics*, 4(4):441–461, November 1962.

[70] N. Srinivas and Kalyanmoy Deb. Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms. *Evolutionary Computation*, 2(3):221–248, 1994.

[71] Rainer M. Storn and Kenneth V. Price. Differential Evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical Report TR-95-012, Berkeley, CA, 1995.

[72] Rainer M. Storn and Kenneth V. Price. Differential Evolution Homepage, http://www.icsi.berkeley.edu/∼storn/code.html, 2007.

[73] E.-G. Talbi. A Taxonomy of Hybrid Metaheuristics. *Journal of Heuristics*, 8(5):541–564, 2002.

[74] V. Torczon. *Multi-directional Search: A Direct Search Algorithm for Parallel Machines*. PhD thesis, Rice University, Houston, Texas, USA, 1989.

[75] Mohamed B. Trabia and Xiao Bin Lu. A Fuzzy Adaptive Simplex Search Optimization Algorithm. *Journal of Mechanical Design*, 123:216–225, 2001.

[76] Alan Mathison Turing. Computing Machinery and Intelligence. *Mind*, 59(236):433–460, 1950.

[77] J. G. van der Corput. Verteilungsfunktionen. *Akademie van Wetenschappen*, 38:813–821, 1935.

[78] David A. Van Veldhuizen and Gary B. Lamont. Multiobjective Evolutionary Algorithm Research: A History and Analysis. Technical Report TR-98-03, Department of Electrical and Computer Engineering, Graduate School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, Ohio, 1998.

[79] David A. Van Veldhuizen and Gary B. Lamont. On Measuring Multiobjective Evolutionary Algorithm Performance. In *2000 Congress on Evolutionary Computation*, pages 204–211, Piscataway, New Jersey, July 2000. IEEE Service Center.

[80] Georges Voronoï. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. Sur quelques propriétés des formes quadratiques positives parfaites. *Journal für die Reine und Angewandte Mathematik*, 133:97–178, 1907.

[81] Georges Voronoï. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. Deuxième mémoire: Recherches sur les parallellotòes primitifs. *Journal für die Reine und Angewandte Mathematik*, 134:198–287, 1908.

[82] Fang Wang and Yuhui Qiu. Multimodal function optimizing by a new hybrid nonlinear simplex search and particle swarm algorithm. In João Gama, Rui Camacho, Pavel Brazdil, Alípio Jorge, and Luís Torgo, editors, *European Conference on Machine Learning*, volume 3720 of *Lecture Notes in Computer Science*, pages 759–766. Springer, 2005.

[83] Xin Yao, Yong Liu, Ko-Hsin Liang, and Guangming Lin. Fast evolutionary algorithms. pages 45–94, 2003.

[84] L. A. Zadeh. Optimality and non-scalar-valued performance criteria. *IEEE Transactions on Automatic Control*, 8(1):59–60, 1963.

[85] Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. *Evolutionary Computing*, 8(2):173–195, 2000.

[86] Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. *Evolutionary Computation*, 8(2):173–195, Summer 2000.

[87] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. Technical Report 103, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zurich, Gloriastrasse 35, CH-8092 Zurich, Switzerland, May 2001.

[88] Eckart Zitzler and Lothar Thiele. Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, November 1999.