ALGORITMOS EVOLUTIVOS PARA OPTIMIZACIÓN MULTIOBJETIVO: UN ESTUDIO COMPARATIVO EN UN AMBIENTE PARALELO ASÍNCRONO

por

Christian Daniel von Lücken Martínez

Trabajo de Tesis desarrollado como parte de los
requerimientos para el grado de
Master

en

Ingeniería de Sistemas

de la
Universidad Nacional de Asunción

Diciembre 2003

Este trabajo está dedicado a mis padres y a mi familia,

sin su amor no lo hubiera logrado.

Agradecimientos

Este trabajo no hubiera sido posible sin el apoyo constante de mis seres queridos, quisiera agradecerles a todos, en especial a Kari por su apoyo y comprensión, por el amor que nos une.

Al profesor Benjamín Barán, su excepcional capacidad para sobrellevar los problemas es fuente de inspiración para todos los que tenemos la suerte de trabajar junto a él. Por todo, gracias profe!.

A la profesora Blanca Troche de Trevisan, directora del Centro Nacional de Computación, por ser la persona que posibilitó que en dicha intitución puedan ser llevados a cabo trabajos de investigación de esta naturaleza.

A Oscar Parra por la ayuda desinteresada y la amistad sincera.

Al profesor Horacio Feliciangelli por la oportunidad de trabajar junto a él.

A Diana Benitez, excelente persona y amiga, gracias.

Quiero agradecer también a mis estimados compañeros y amigos del grupo de investigación, con quienes pasé gratos momentos que hicieron más llevaderos los imprevistos y las contrariedades muy habituales en este tipo de trabajo.

A todos mis profesores, compañeros y amigos de la facultad con quienes compartí estos años de formación.

A mis primos, Rodrigo y Luis Fernando. Luis, sé que me estás mirando.

Finalmente a mis abuelos, por cuidarme.

Índice general

Índice de algoritmos

Índice de Tablas

Capítulo 1

Introducción

Los problemas del mundo real usualmente requieren la búsqueda de soluciones que satisfagan de manera simultánea múltiples criterios de desempeño u objetivos, los cuales pueden ser contradictorios y no conmensurables [64, 67]. Si existe la posibilidad de combinar los diferentes objetivos de un problema y se conoce la mejor manera de hacerlo, entonces se puede considerar que existe un único objetivo a optimizar. En este caso, para obtener *la solución* del problema basta con encontrar el mínimo o el máximo de una única función que resume todos los objetivos que se quieren satisfacer. Sin embargo, lo usual es que no se conozca de que manera combinar los diferentes objetivos, o sea inadecuado, cuando no imposible hacerlo. Entonces, se dice que el problema es un Problema de Optimización Multiobjetivo (*Multiobjective Optimization Problem - MOP*) [25, 50, 64, 86, 101].

En los problemas de optimización multiobjetivo con objetivos contradictorios no existe una solución única que pueda ser considerada la mejor, sino un conjunto de soluciones que representan los mejores compromisos entre los objetivos [64, 67]. Éste es un conjunto de soluciones óptimas en el sentido que cada una es mejor que las otras en algún objetivo, pero ninguna es mejor que otra en todos los objetivos simultáneamente. Dicho conjunto es llamado conjunto de soluciones Pareto-óptimas y sus correspondientes vectores en el espacio objetivo constituyen el denominado Frente Pareto [18, 50].

Los Algoritmos Evolutivos (*Evolutionary Algorithms - EAs*) [4] han demostrado ser especialmente adecuados para la optimización multiobjetivo [20, 21, 100]. Desde la década de los 80s se han desarrollado varios EAs, capaces de buscar múltiples soluciones Pareto-óptimas en forma simultánea en una sola corrida [1, 17, 18, 19, 29, 42]. Con el paso de los años, la Optimización Evolutiva Multiobjetivo (EMOO - *Evolutionary Multiobjective Optimization*) se ha establecido como un área importante de investigación recibiendo cada vez mayor atención [86].

En la actualidad, existe un gran número de Algoritmos Evolutivos para Optimización Multiobjetivo (*Multiobjective Evolutionary Algorithms - MOEA*) [24, 86]. Con el uso cada vez más extendido de estos algoritmos en problemas reales de optimización, se hace necesario mejorar el desempeño de los mismos [5, 37, 92]. Por tanto, para asegurar la aplicabilidad de la técnica de optimización evolutiva multiobjetivo a problemas de complejidad creciente, es necesario mejorar tanto en la efectividad como en la eficiencia de los métodos evolutivos. Para ello, una alternativa es la incorporación de conceptos de paralelismo al diseño de estos algoritmos [37, 88, 91].

Los conceptos de paralelismo han estado presentes en la literatura referente a EAs prácticamente desde sus origines. En [11, 12, 13] se presenta una revisión de varios modelos de paralelización para algoritmos evolutivos mono-objetivo. Estos modelos apenas han sido integrados al campo de la optimización evolutiva multiobjetivo [86, 88, 89]. Siendo el área de paralelización de algoritmos evolutivos multiobjetivo un área de gran importancia y reciente interés, es necesario determinar las ventajas y desventajas existentes en el desarrollo y aplicación de algoritmos evolutivos multiobjetivo paralelos (*parallel Multiobjective Evolutionary Algorithms - pMOEAs*) [88, 89].

A fin de determinar la efectividad de la técnica, como en [2, 3, 86, 87, 97, 98, 99, 103], en el presente trabajo se realiza una comparación de distintos pMOEAs utilizando diferentes métricas experimentales propuestas para medir el desempeño de MOEAs [36, 86, 103]. Para ello, se han realizado implementaciones paralelas de seis algoritmos evolutivos para optimización multiobjetivo, aplicados

a funciones de prueba de distinta dificultad [98]. Además, debido a la importancia de un problema real de optimización previamente estudiado: *el calendarizado óptimo del encendido de bombas en estaciones de bombeo de agua*, en el presente trabajo se utiliza dicho problema como paradigma principal [80, 92].

Así, los objetivos de este trabajo son:

- Presentar los conceptos más importantes de EMOO.
- Reportar la bibliografía publicada en el área de EMOO.
- Presentar una comparación experimental entre diferentes pMOEAs.
- Mejorar la habilidad de búsqueda de distintos MOEAs utilizando conceptos de paralelismo, en especial para el problema de calendarizado de estaciones de bombeo de agua.
- Desarrollar fundamentos empíricos y teóricos para ayudar a decidir cual representa una mejor opción con respecto a las características de un problema dado.

El presente trabajo está organizado de acuerdo al siguiente esquema:

- El capítulo 2 presenta los conceptos de computación evolutiva utilizados en el transcurso del trabajo, haciendo especial referencia a los algoritmos genéticos, sus conceptos, las características que los hacen más atractivos que otros métodos para la resolución de problemas, su funcionamiento, y las razones de su éxito.
- El capítulo 3 presenta los conceptos claves relacionados a optimización multiobjetivo.
- En el capítulo 4 presenta una descripción de los distintos algoritmos evolutivos multiobjetivo utilizados en este trabajo.

- El capítulo 5 presenta conceptos de computación paralela y la manera en que estos son incorporados a la optimización evolutiva multiobjetivo.

- El capítulo 6 presenta las distintas métricas utilizadas, así como los resultados de la utilización de un modelo de paralelización determinado sobre distintas funciones de prueba.

- El capítulo 7 presenta el problema de calendarizado óptimo del encendido de bombas en una estación de bombeo de agua y una comparación de los resultados obtenidos por distintos pMOEAs para la resolución de éste problema.

- Finalmente se presentan las conclusiones y los trabajos futuros.

Capítulo 2

Computación Evolutiva

Uno de los temas centrales de las ciencias de la computación es cómo lograr que una computadora encuentre soluciones para cualquier problema, a partir de una descripción de alto nivel del mismo, sin tener que programarla en forma explícita. Esto es, cómo construir sistemas dotados de un comportamiento inteligente y que no sólo por su velocidad o precisión sean capaces de resolver problemas al descubrir por sí mismos, eficaz y eficientemente, soluciones óptimas para la amplia gama de problemas posibles [9].

Si bien el concepto de inteligencia es algo difícil de definir, se puede decir que un sistema es inteligente si es capaz de mejorar su desempeño o mantener un nivel aceptable en presencia de incertidumbre. Atributos principales de estos sistemas son la capacidad de aprendizaje, de adaptación, la tolerancia a fallas y la auto-organización [56]. Estas características, propias de los seres vivos, raramente se encuentran presentes en los sistemas artificiales, incluso en los más avanzados. La inteligencia artificial (IA) intenta construir sistemas con un comportamiento inteligente [48, 72].

Para encarar su ambicioso objetivo, el núcleo de la investigación en inteligencia artificial se divide, básicamente, en representación del conocimiento, búsquedas y aplicaciones de estas ideas [48]. La representación del conocimiento estudia el problema de encontrar un lenguaje en el cual codificar el

conocimiento de forma tal que la máquina pueda usarlo. Mientras que el área de búsqueda, trata con la resolución de problemas donde la técnica básica utilizada implica examinar un gran número de posibilidades en la búsqueda de la solución.

La importancia de la búsqueda para la IA radica en que muchos problemas del mundo real que implican esfuerzo intelectual, pueden plantearse en forma directa como problemas de búsqueda. Un ejemplo clásico es el juego de ajedrez, donde dada una posición se busca entre una secuencia de movidas con el fin de determinar la mejor. Además, distintos problemas pueden ser planteados adecuadamente como problemas de búsqueda. El primero en establecer una relación entre inteligencia de máquina y la búsqueda fue Allan Turing, cuando a fines de la década del 40 identificó varios enfoques que podrían ser utilizados para crear programas inteligentes a partir de la búsqueda [85].

Para poder construir las cajas negras solucionadoras de problemas, se precisa contar con algoritmos que requieran un mínimo de información acerca del espacio de búsqueda explorado. Estos algoritmos son conocidos usualmente como métodos de búsqueda débiles (*weak methods*), debido a que la información sobre el dominio del problema con que cuentan es débil [72].

Los métodos débiles tienen en general la dificultad que para muchos problemas, el espacio de búsqueda crece exponencialmente con respecto al tamaño de la instancia considerada, siendo imprácticto examinar las distintas posibilidades sin incorporar información adicional. Entonces, es deseable la utilización de métodos que además de precisar poco conocimiento a priori, sean capaces de generar por sí mismos información que permita orientar su búsqueda a fin de comportarse de forma eficiente en espacios de búsqueda complejos y de alta dimensionalidad. Para el desarrollo de tales métodos, se ha encontrado en la naturaleza y especialmente en el proceso evolutivo, una importante fuente de inspiración. De hecho, la evolución en los entornos naturales puede verse como un proceso de búsqueda entre un gran número de posibles combinaciones de genes. Como resultado de dicha búsqueda se obtienen aquellas secuencias que permiten a los organismos sobrevivir y reproducirse en sus diferentes

entornos, adaptándose a los cambios que se producen en el ambiente. Así, a partir de las nociones provenientes de la evolución biológica surge la computación evolutiva (*Evolutionary Computation - EC*) como una forma de simular el proceso evolutivo para la resolución de distintos problemas computacionales [4].

Las primeras ideas de búsquedas genéticas o evolutivas aparecieron a principios de los 50's como una curiosidad científica. En los 60's se desarrollaron las bases de los métodos principales de la computación evolutiva: los Algoritmos Genéticos (*Genetic Algorithm - GA*) [10, 51, 52], las Estrategias de Evolución (*Evolution Strategies - ES*) [69, 79] y la Programación Evolutiva (*Evolution Programming - EP*) [41]. La falta de recursos computacionales suficientes para la implementación de estos métodos en problemas reales los llevó a un periodo de rechazo y poco interés. En la década del 80, con el aumento del poder de cómputo, estos algoritmos recibieron mayor atención como métodos para la resolución de problemas. Sin embargo, recién a partir de la década del noventa estos métodos se establecieron como alternativas prácticas cuando fueron utilizados con éxito para la resolución de una amplia variedad de problemas de optimización y búsqueda de gran complejidad [4, 49].

Usualmente, los distintos métodos de la computación evolutiva son llamados colectivamente Algoritmos Evolutivos (*Evolutionary Algorithms - EA*). Si bien cada enfoque tiene sus características particulares, en los últimos años se han propuesto un gran número de EAs que las combinan, quedando difuso el límite entre éstos. Siendo los algoritmos genéticos los EAs que han recibido mayor atención de la comunidad científica en los últimos años y que comparten ciertas características con los otros enfoques, usualmente se utiliza el término algoritmo genético como sinónimo de algoritmo evolutivo a menos que se indique otra cosa [49].

En el presente capítulo presentamos una descripción de los principios de funcionamiento de la técnica de los algoritmos evolutivos con énfasis en los algoritmos genéticos tradicionales [50, 52], como ejemplo ilustrativo de los EAs.

2.1. Algoritmos Evolutivos

Como se ha dicho, el proceso de evolución puede ser visto como una búsqueda entre el conjunto enorme de posibles secuencias de genes de aquellas que correspondan a los mejores individuos. Esta búsqueda que la naturaleza realiza es robusta, en el sentido que es capaz de localizar eficaz y eficientemente individuos de mejor adaptación en un espacio de búsqueda enorme y cambiante.

Las reglas de la evolución son notablemente simples. Las especies evolucionan por medio de la selección natural, esto es, la conservación de las diferencias y variaciones individualmente favorables y la destrucción de las que son perjudiciales. Los individuos que tienen ventaja sobre otros, por ligera que sea, tendrán más probabilidades de sobrevivir y procrear su especie [26].

Los hijos son semejantes a sus padres, por lo que cada nueva generación tiene individuos semejantes a los mejores individuos de la generación anterior. Además del proceso de selección natural, en la naturaleza ocasionalmente se producen mutaciones al azar. Algunas de estas mutaciones conducirán a nuevos y mejores individuos, de esta manera, alteraciones aleatorias serán introducidas y luego propagadas a generaciones futuras. Por el contrario, las características negativas introducidas por la mutación tenderán a desaparecer pues conducen a individuos pobremente adaptados. Si las modificaciones del ambiente son menores, las especies irán evolucionando gradualmente con éste; sin embargo, es probable que un súbito cambio de ambiente provoque la desaparición de especies enteras y el nacimiento de otras nuevas, capaces de sobrevivir en el nuevo entorno.

Los algoritmos evolutivos son algoritmos de búsqueda y optimización que simulando los mecanismos esenciales de la evolución Darwiniana, intentan reproducir las características de robustez y simplicidad existentes en la naturaleza para evolucionar hacia mejores soluciones en problemas computacionales [50].

Estos algoritmos trabajan con un conjunto de posibles soluciones para el problema que se intenta resolver. Cada una de estas soluciones representa un individuo y todo el conjunto representa una

población. La información relativa a las características de los individuos en poblaciones naturales viene dada en los cromosomas, los cuales son responsables de la carga hereditaria. Por ello, cuando se trabaja con EAs, las soluciones del problema considerado se codifican en forma de cadenas semejantes a las cadenas de genes que conforman los cromosomas.

En la naturaleza cada individuo posee unas características particulares, que miden de alguna manera su capacidad de adaptabilidad con respecto a otros individuos de la misma especie en un entorno dado. Del mismo modo, en los EAs, a cada solución de la población artificial se le asigna un valor de adaptabilidad (*fitness*) de acuerdo a que tan buena sea como solución del problema en cuestión. Dicho valor servirá para que un proceso aleatorio de selección artificial, que favorece a los individuos con mejor adaptabilidad, tome pares de individuos de la población para ser sometidos a un operador de cruzamiento. El operador de cruzamiento implementa alguna forma de intercambio entre las cadenas que conforman los individuos seleccionados (*padres*), para generar nuevas soluciones (*hijos*) las cuales formarán una nueva población.

Los procedimientos de asignación de *fitness*, selección y cruzamiento se repiten de forma iterativa hasta que algún criterio de parada se cumpla. A cada iteración se le denomina comúnmente generación. Entonces, se dice que un nuevo conjunto de soluciones se crea en cada generación por medio de la selección de buenos individuos de la generación actual y el cruzamiento de los mismos.

Los miembros de cada nueva generación creada a partir de la selección y el cruzamiento tendrán características similares a las de los mejores individuos (soluciones) de la generación anterior. Sin embargo, ciertas características importantes pueden perderse en este proceso. Un operador de mutación que altera en forma aleatoria una solución tiene a su cargo recuperar la información que se pueda perder durante el proceso simulado de evolución natural, introduciendo una búsqueda aleatoria en el espacio de soluciones. Este proceso análogo al llevado a cabo en entornos naturales, proporciona a los EAs las características necesarias para un comportamiento robusto que le permite ser utilizado para

un amplio rango de problemas de optimización de funciones, optimización combinatoria, sistemas de aprendizajes de lógica difusa, aprendizaje de redes neuronales, entre otros [15, 29, 86, 100].

2.2. Métodos tradicionales de búsqueda, algoritmos evolutivos y robustez

Existe una gran variedad de métodos tradicionales de búsqueda y optimización, los cuales se pueden clasificar básicamente en tres categorías: enumerativos, determinísticos y aleatorios. Si bien los algoritmos enumerativos son algoritmos determinísticos, se hace una distinción para indicar que no utiliza ninguna heurística [86].

Los esquemas enumerativos son probablemente los más simples. Básicamente se evalúa cada solución posible de un espacio de búsqueda finito. En espacios de búsqueda de gran tamaño, esta técnica es ineficiente e incluso imposible de aplicar, por lo que se necesita limitar de alguna manera el espacio de exploración. Los algoritmos determinísticos intentan realizar tal reducción incorporando información específica del dominio del problema. Algunos algoritmos determinísticos son:

- Los algoritmos voraces (*greedy*), que realizan elecciones localmente óptimas asumiendo que subsoluciones óptimas son siempre parte de la solución global óptima. Estos algoritmos son adecuados sólo cuando este caso se cumple .

- Los algoritmos de ascenso de cimas (*hill-climbing*), que buscan en la dirección de la mejora a partir de la posición actual. Estos algoritmos funcionan bien para funciones unimodales, pero la presencia de óptimos locales o llanos en el espacio de búsqueda reducen la efectividad del algoritmo.

- Los algoritmos de búsqueda en profundidad, que son algoritmos de búsqueda ciega donde la

búsqueda se realiza de forma independiente a la ubicación de la solución. Esta expande un nodo, genera todos sus sucesores, expande su sucesor y así sucesivamente. Si la búsqueda alcanza el nivel inferior, la búsqueda se reinicia en el nivel anterior.

- La búsqueda en amplitud es también ciega y difiere de la búsqueda en profundidad en que explora el grafo un nivel a la vez.

- La búsqueda del mejor primero (*best-first search*) utiliza información heurística para asignar valores numéricos a los nodos, aquellos con mejor valor son examinados primero. Los métodos de $A^\star$, $Z^\star$ y otros [48] son variantes de este método donde la selección del nodo a expandir se realiza de acuerdo a que tan promisorio es, así como en el costo total incurrido para alcanzarlo.

Muchos problemas tienen alta dimensionalidad, son discontinuos, multimodales y/o NP-completos. Los problemas que poseen una o más de estas características son denominados irregulares. Si bien los métodos determinísticos han sido utilizados exitosamente en la resolución de una gran variedad de problemas, son usualmente inefectivos cuando se aplican a problemas irregulares.

La búsqueda aleatoria más simple analiza de forma aleatoria una cierta cantidad de soluciones en busca del óptimo. La caminata aleatoria es muy similar, pero la selección del siguiente elemento a evaluar se realiza considerando la última solución evaluada como punto de partida. Como los métodos enumerativos, para muchos problemas estas estrategias no son eficientes puesto que no consideran conocimiento acerca del dominio del problema. En general, las búsquedas aleatorias se comportan de forma similar a los métodos enumerativos en espacios de búsqueda de gran dimensión.

La Tabla 2.1 resume las características principales de estas categorías de algoritmos y sus desventajas con respecto a la robustez. En dicha tabla, los métodos determinísticos se han subclasificado en métodos directos e indirectos [50]. Aquí, un método es llamado directo si utiliza la función objetivo en forma directa, mientras se dice que un método es indirecto si precisa información adicional a la fun-

	Determinísticos		*Enumerativos*	*Aleatorios*
	Indirectos	*Directos*		
Características	Buscan óptimos al resolver un conjunto de funciones no lineales que resultan de hacer el gradiente de la función objetivo igual a cero.	Buscan óptimos locales saltando sobre la función y moviéndose en dirección relativa al gradiente local.	Buscan soluciones recorriendo el espacio de búsqueda y analizando cada solución, una a la vez.	Buscan soluciones recorriendo en forma aleatoria el espacio de búsqueda.
Desventajas	•Suposiciones restrictivas de continuidad, existencia de derivadas, etc. para la función objetivo. •Necesidad de un conocimiento profundo del problema en cuestión. •Buscan la solución a partir de un punto dado. •Poseen alcance local, el óptimo que buscan son los mejores en las cercanías del punto actual, por lo que pueden con facilidad perderse el evento principal.		•Muchos espacios de búsqueda son simplemente demasiado grandes para buscar punto por punto. •No utilizan información relativa al espacio de búsqueda. •Buscan soluciones sin tener en cuenta información sobre el espacio ya recorrido.	

Tabla 2.1. Características principales de los métodos de búsqueda y optimización tradicionales.

ción objetivo. Un análisis más detallado puede encontrarse en [50, 86]. En síntesis, se puede concluir que existe un gran número de problemas que no pueden ser resueltos en forma efectiva utilizando los enfoques tradicionales de búsqueda.

De acuerdo al *no-free-lunch theorem* (*NFL*), no existe ningún algoritmo para resolver todos los problemas de optimización que sea en promedio superior a cualquier buen competidor [94, 95]. Sin embargo, los algoritmos evolutivos han demostrado poseer características que les permiten presentar un mejor comportamiento que el demostrado por otros métodos para resolver un gran número de problemas reales donde las técnicas clásicas no pueden ser aplicados, aunque como consecuencia, se comportan peor para otra clase de problemas. En particular, los métodos clásicos son muy eficientes para la resolución de problemas lineales o cuadráticos, fuertemente convexos y unimodales. Por otra parte, los algoritmos evolutivos pueden ser utilizados en problemas discontinuos, no diferenciables, multimodales, ruidosos y en otras superficies de búsqueda poco convencionales donde la búsqueda por métodos clásicos es impracticable. Entonces, su efectividad se extiende a un número mayor de

aplicaciones, por supuesto, con una pérdida de eficiencia cuando se aplican a problemas para los cuales se tiene un método específico para resolverlo. En este sentido, se puede decir que los EAs son más robustos que otras alternativas. Las características generales que hacen a los EAs diferentes a los métodos tradicionales en cuanto a la robustez son:

1. Trabajan con poblaciones de individuos donde cada individuo representa (o codifica) un punto en el espacio de soluciones potenciales para un problema dado. Esto implica que trabajan con muchos puntos reduciendo de ésta forma la posibilidad de perderse en óptimos locales.

2. Las soluciones se generan de forma iterativa mediante la utilización de operadores aleatorios que intentan modelar los procesos esenciales que intervienen en la evolución natural. Existen tres operadores evolutivos principales: cruzamiento, mutación y selección.

3. A fin de evaluar los individuos se utiliza información de su función objetivo en forma directa para asignarle un valor de adaptación que permite determinar si un individuo es mejor que otro.

La diferencia básica existente entre los diferentes enfoques de la computación evolutiva radica en la manera en que los principios evolutivos son utilizados para obtener las características anteriores. Sin pretender analizar en profundidad las diferencias entre los diferentes modelos evolutivos principales se tiene que:

- Los algoritmos genéticos enfatizan el operador de cruzamiento como el operador de búsqueda más importante y aplican la mutación con una pequeña probabilidad como un operador de segundo plano. Los GAs también utilizan un operador de selección probabilístico y usualmente utilizan una representación binaria de los individuos.

- Las estrategias de evolución utilizan mutaciones con distribución normal para modificar vectores de valores reales enfatizando la mutación y el cruzamiento como sus operadores principales

para la exploración del espacio de búsqueda. El operador de selección es determinístico y las poblaciones padre e hijo usualmente difieren en tamaño.

- La programación evolutiva enfatiza la mutación y no incorpora la recombinación de individuos. Al igual que las estrategias de evolución, utilizan distribución normal para su operador de mutación. El operador de selección es probabilístico, y aunque en la actualidad la mayor parte de sus aplicaciones son para búsquedas en espacios de vectores de valor real, el algoritmo fue desarrollado en un principio para evolucionar máquinas de estado finito.

Un análisis detallado sobre las similaridades y diferencias existentes entre estos métodos puede encontrarse en [4].

2.3. Codificación y función de adaptación

En general, para resolver un problema utilizando una computadora, primeramente buscamos la secuencia de pasos que deben ejecutarse para resolverlo (el algoritmo). Luego, utilizando algún lenguaje de programación, se codifica esta secuencia de pasos en un conjunto de instrucciones que se han de ejecutar. Cuando se trabaja con algoritmos evolutivos, sin embargo, la preocupación principal no es acerca de qué pasos conducirán a la solución del problema sino más bien en la manera de representar las soluciones. Es el algoritmo evolutivo el que se encargará de evolucionar la solución. Así, la computación evolutiva representa un cambio radical con respecto a la manera tradicional de enfrentar los problemas computacionales.

Para resolver un problema utilizando técnicas de computación evolutiva, las posibles soluciones son codificadas de alguna manera similar a la estructura de las cadenas de genes que conforman los cromosomas. Los parámetros del problema se codifican ya sea como una cadena de números binarios [50], números reales [74], letras del alfabeto, listas, etc. Cada parámetro puede tener una codificación

independiente de acuerdo a la conveniencia. En analogía con la naturaleza, cada parámetro del problema codificado representa un gen y considerados en conjunto, forman una cadena de genes o cromosoma. Así mismo, los elementos en el conjunto de parámetros son denominados usualmente genotipos, mientras que los elementos en el espacio objetivo suelen denominarse fenotipos. Las definiciones 2.1 al 2.5 formalizan estas ideas [4].

Definición 2.1. *Codificación:* Sea $\mathcal{S}$ un espacio de búsqueda, i.e. una colección de objetos sobre el cual se realizará la búsqueda. Sean $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n$ conjuntos finitos arbitrarios y sea:

$$I = \mathcal{A}_1 \times \mathcal{A}_2, \ldots, \times \mathcal{A}_n$$

Finalmente sea

$$\mathrm{g} : I \to \mathcal{S}$$

una función que mapea vectores en I a soluciones en el espacio de búsqueda. Se dice que el par $(I, \mathrm{g} : I \to \mathcal{S})$ es una codificación o representación de $\mathcal{S}$. El conjunto I es denominado espacio de codificación, la función g es llamada la función de mapeo. Entonces, n es el número de variables utilizadas para mapear una solución de $\mathcal{S}$ en I.

Definición 2.2. *Alelo:* Los conjuntos $\mathcal{A}_\mathrm{i}$, $\mathrm{i} \in \{1, \ldots, n\}$, a partir de los cuales se compone I en la definición 2.1, es llamado el conjunto de alelos y sus miembros alelos.

Definición 2.3. *Cromosoma:* Sea la codificación $(I, \mathrm{g} : I \to \mathcal{S})$, los miembros de I son llamados cromosomas y menos comúnmente genomas o genotipos. Un cromosoma $\mathbf{x} \in I$ puede ser escrito como:

$$(x_1, x_2, \ldots, x_n) \in \mathcal{A}_1 \times \mathcal{A}_2, \ldots, \times, \mathcal{A}_n$$

o como una cadena $x_1 x_2 \ldots x_n$.

Definición 2.4. *Gen:* Los componentes x_i de **x**, cuando se tratan como variables, son conocidos como genes, tal que el i-ésimo gen toma sus valores del conjunto de alelos $\mathcal{A}_i$.

Definición 2.5. *Locus:* La posición i de un gen en el cromosoma es conocida como su *locus*.

Como ejemplo, considerando el espacio de búsqueda $\mathcal{S} = \{0,1,2,\ldots,9\}$ y un espacio de representación $I = \mathcal{A}_1 \times \mathcal{A}_2$, donde $\mathcal{A}_1 = \{0,1\}$ y $\mathcal{A}_2 = \{0,1,2,3,4\}$. Una función de mapeo $g : I \to \mathcal{S}$ puede definirse adecuadamente como:

$$g(x_1,x_2) = 5 \cdot x_1 + x_2$$

donde x_1 toma sus valores del conjunto de alelos $\mathcal{A}_1$ y x_2 toma sus valores del conjunto de alelos $\mathcal{A}_2$. Entonces, para la codificación $(I, g : I \to \mathcal{S})$, un cromosoma $\mathbf{x} = x_1x_2 = 13$ en I representa a la solución 8 en $\mathcal{S}$. Además, se dice que el gen con *locus* 1 (x_1), tiene como valor el alelo 1 y el gen con *locus* 2 tiene como valor el alelo 3 [4].

En muchas representaciones, todos los genes tienen la misma cardinalidad y comparten un mismo conjunto de alelos. En la mayoría de las aplicaciones de GAs, las variables de decisión son codificadas utilizando representaciones binarias, en las cuales todos los genes tienen los alelos 0 o 1. Esto es, $\mathcal{A}_i = \{0,1\}$ para todo $i \in \{1,\ldots,n\}$.

La representación binaria es adecuada para los problemas llamados problemas de optimización combinatoria seudo-Boléanos de la forma $f : \{0,1\}^n \to \mathbb{R}$. En éstos problemas, el espacio de búsqueda ($\{0,1\}^n$) puede ser representado en forma directa por cadenas binarias de longitud $l = n$. Es decir, para este caso se tiene $\mathcal{S} = I$. Algunos ejemplos de tales problemas son el problema del conjunto independiente máximo en grafos [58] o el problema de la mochila [57], los cuales pueden ser representados por vectores binarios simplemente incluyendo (o excluyendo) un vértice o item i en (de) una solución candidata cuando la entrada correspondiente se hace $x_i = 1$ ($x_i = 0$).

Esta representación también ha sido muy utilizada en el caso de problemas $f : S \to \mathbb{R}$ donde el espacio de búsqueda S es fundamentalmente distinto del espacio vectorial binario $\{0,1\}^l$. Uno de los ejemplos más prominentes de esto está dado por la aplicación de algoritmos genéticos para problemas de optimización de parámetros continuos de la forma $f : \mathbb{R}^n \to \mathbb{R}$ [27]. Los mecanismos para codificar y decodificar entre los dos espacios diferentes $\{0,1\}^l$ y $\mathbb{R}^n$ restringen el espacio continuo a intervalos finitos $[u_i; v_i]$ para cada variable $x_i \in \mathbb{R}$, dividiendo el vector binario de tamaño l en n segmentos. Por lo general estos segmentos tienen igual longitud l_x, tal que $l = n \cdot l_x$. Siendo a_j el valor (0 o 1) específico en la posición j de la cadena binaria, se interpreta el subsegmento $(a_{(i-1)\cdot l_x+1}, \ldots, a_{i\cdot l_x})(i = 1, \ldots, n)$ como la codificación binaria de la variable x_i. Entonces, la decodificación del i-ésimo gen representado en forma binaria con una subcadena de tamaño $l_i = l_x$ se realiza de acuerdo a la función de decodificación binaria $\Gamma^i : \{0,1\}^{l_x} \to [u_i; v_i]$, donde [4]:

$$\Gamma^i(a_{i\cdot 1}, \ldots, a_{i\cdot l_x}) = u_i + \frac{v_i - u_i}{2^{l_x} - 1}\left(\sum_{j=0}^{l_x-1} a_{i\cdot(l_x-j)} 2^j\right) \tag{2.1}$$

Por ejemplo, una variable real x_1 en el rango $[0{,}5\ ;\ 16]$ puede ser codificada en cadenas de 5 bits. Entonces, utilizando la ecuación 2.1, las cadenas (00000) y (11111) representan los valores reales 0,5 y 16 respectivamente, mientras que cualquiera de las otras 30 cadenas posibles representan una solución en el intervalo $(0{,}5\ ;\ 16)$:

$$\Gamma^1(00000) = 0{,}5 + \frac{16 - 0{,}5}{2^5 - 1} \cdot 0 = 0{,}5$$
$$\Gamma^1(00001) = 0{,}5 + \frac{16 - 0{,}5}{2^5 - 1} \cdot (1 \cdot 2^0) = 1{,}0$$
$$\Gamma^1(00010) = 0{,}5 + \frac{16 - 0{,}5}{2^5 - 1} \cdot (1 \cdot 2^1 + 0 \cdot 2^0) = 1{,}5$$
$$\Gamma^1(00011) = 0{,}5 + \frac{16 - 0{,}5}{2^5 - 1} \cdot (1 \cdot 2^1 + 1 \cdot 2^0) = 2{,}0$$
$$\vdots$$
$$\Gamma^1(11111) = 0{,}5 + \frac{16 - 0{,}5}{2^5 - 1} \cdot 31 = 16{,}0$$

Luego de definir la codificación, se debe establecer una función de adaptación para cada problema a ser resuelto. Como se dijo, un cromosoma particular representa una solución al problema considerado. Idealmente, a partir de un cromosoma dado la función de adaptación debe retornar un valor proporcional a la utilidad de la solución representada por tal cromosoma. Esto podría no parecer un problema, debido a que cada problema de optimización posee una métrica (o métricas) relevante para la búsqueda [50].

Para ciertos problemas, particularmente con la optimización de funciones, es fácil determinar que función de adaptación debería ser utilizada, sin embargo, este no es el caso usual en problemas prácticos. La función de adaptación se puede definir formalmente como:

Definición 2.6. *Función de adaptación:* Sea el espacio de búsqueda $\mathcal{S}$ y sea la codificación $(I, \mathsf{g} : I \to \mathcal{S})$. Entonces, una función de adaptación es cualquier función

$$\mathsf{f} : I \to \mathbb{R} \tag{2.2}$$

tal que, cumpla con la propiedad que

$$\forall \mathbf{x}, \mathbf{x}' \in I, \mathsf{f}(\mathbf{x}) > \mathsf{f}(\mathbf{x}'), \text{ si } \mathsf{g}(\mathbf{x}) \text{ se considera mejor que } \mathsf{g}(\mathbf{x}') \text{ en } \mathcal{S} \tag{2.3}$$

Es decir, dado cualquier par de cromosomas $\mathbf{x}$, $\mathbf{x}'$, al evaluar la función de adaptabilidad sobre estos, tendrá un valor de adaptación mayor a aquel que represente la mejor solución en el espacio de búsqueda considerado.

Definición 2.7. *Individuo:* Sea la codificación $(I, \mathsf{g} : I \to \mathcal{S})$, y sea el cromosoma $\mathbf{x} \in I$, se utiliza el término individuo para referenciar una estructura de datos compuesta por un cromosoma y otras propiedades, como por ejemplo su valor de adaptabilidad f.

Definición 2.8. *Población:* Una población es una estructura de datos que representa un grupo de individuos. Generalmente se representa a una población por medio de un vector de individuos.

Para continuar, resulta conveniente introducir algunas notaciones. El conjunto de soluciones o población utilizado por los GAs para simular el proceso evolutivo se representa en este trabajo con P. Como se verá posteriormente, esta población cambia, generación a generación, conforme el proceso simulado de evolución avanza. Para representar esto se denota como $P(t)$ a una población P en una generación particular t. Por simplicidad, esta notación se utilizará sólo cuando es preciso señalar el número t de generaciones transcurrido desde el inicio de la ejecución del algoritmo. En otros casos, se utilizará sólo P. A un individuo que se encuentra en la posición cualquiera i de la población $P(t)$ se le representa, de acuerdo a la conveniencia, ya bien utilizando un subíndice como $P_i(t)$ o utilizando la notación $P[i](t)$. Igualmente, se utiliza sólo P_i o $P[i]$ cuando no resulta necesario indicar el número de generación. Las propiedades de los individuos varían de acuerdo al algoritmo y la representación considerada. Como un ejemplo de la notación utilizada, el valor de adaptación de un individuo se representa con el subíndice $fitness$. Así, el valor de adaptación del individuo en la posición i de la población P, puede representarse como: $P_{i_{fitness}}$ o más convenientemente como $P[i]_{fitness}$; mientras que el respectivo cromosoma como: $P[i]_{chrom}$. El tamaño de una población genética P se representa con N. Notación referente a otras propiedades será incluída en el presente trabajo conforme fuera necesario.

2.4. Operadores evolutivos

Los individuos en las poblaciones intercambian información por medio de operadores evolutivos. Existen tres operadores genéticos principales: selección, cruzamiento y mutación. Las siguientes secciones explican en forma breve cada uno de ellos.

2.4.1. Selección

La selección es un proceso por medio del cual los individuos son escogidos de la población de acuerdo al valor de su función de adaptación para someterse a la acción futura de otros operadores. Entre los métodos de selección más utilizados están la selección vía ruleta (*roulette-whell selection* - RW) y la selección por torneo (*tournament selection* - TS) [50].

El procedimiento de selección por medio de ruleta es una de las estrategias de selección más simples. Este procedimiento se muestra en el Algoritmo 1. En la selección por ruleta, la probabilidad de que un individuo sea seleccionado se calcula en forma directa a partir del valor de adaptación del individuo. Básicamente consiste en la construcción de una ruleta donde a cada individuo se le asigna una fracción de la misma en proporción directa a su adaptabilidad. Luego se selecciona en forma aleatoria una posición en la ruleta, el individuo al cual le corresponda dicha posición es el que será seleccionado. Entonces la probabilidad de selección del individuo $P[i]$ como padre es:

$$p_s(i) = \frac{P[i]_{fitness}}{\sum_{j=1}^{N} P[j]_{fitness}} \tag{2.4}$$

Otro método de selección comúnmente utilizado es el método de selección por torneo. En la selección por torneo un número dado de individuos de la población es preseleccionado en forma aleatoria. Estos individuos se comparan unos con otros de acuerdo a su valor de adaptación de una manera similar a la efectuada en competencias deportivas. Aquel con mejor valor de adaptación es seleccionado.

Algoritmo 1 Algoritmo de selección por ruleta.

$sum = \sum_{j=1}^{N} P[j]_{fitness}$
$r = rnd(0, sum)$ // *rnd* retorna un número aleatorio entre 0 y *sum*
for $j = 1$ to N **do**
  $r = r - P[j]_{fitness}$
  if $r \leq 0$ **then**
    return $P[j]$
  end if
end for

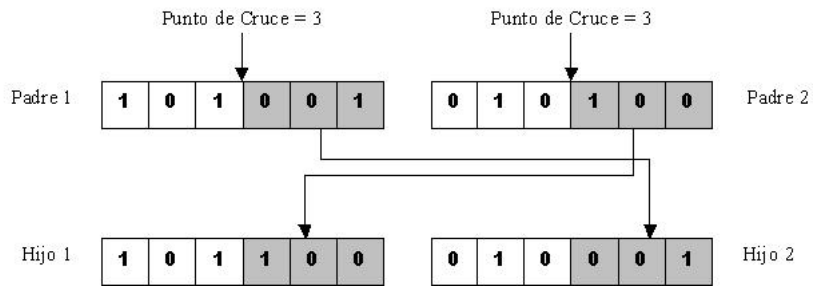

Figura 2.1. Funcionamiento del operador de cruzamiento de un sólo punto.

Cuando el número de individuos elegidos es dos, al TS se le llama selección por torneo binario. Ajustando el tamaño del torneo se puede obtener algún control sobre la presión de selección y por tanto de la velocidad de convergencia. El menor tamaño de torneo (torneo binario) exhibe una convergencia más lenta que cualquier tamaño de torneo mayor.

2.4.2. Cruzamiento

El proceso de cruzamiento se inicia a partir de la obtención, utilizando algún operador de selección, de los individuos que serán sometidos al operador de cruzamiento que se encargará de intercambiar los componentes de los individuos seleccionados para producir nuevas soluciones. Así, el operador de cruzamiento se encarga de la transferencia por herencia de las características de los mejores individuos de una generación a la siguiente, siendo de esta manera la contraparte artificial de la recombinación sexual.

Existen varias opciones para implementar el operador de cruzamiento siendo el cruzamiento de un sólo punto la forma más común de implementar este operador cuando se trabaja con una codificación de cadenas de bits. Este tipo de cruzamiento requiere la selección de dos cadenas como padres, luego, se selecciona un punto de cruzamiento, tal que dicho punto se encuentre entre 1 y $(l-1)$, donde l es la longitud de la cadena. Los padres son cortados en este punto para producir cuatro subcadenas, que se intercambian para generar dos hijos. La figura 2.1 muestra este procedimiento.

2.4.3. Mutación

En los GAs la mutación es utilizada como un operador de segundo plano cuyo propósito es la exploración aleatoria de nuevas porciones del espacio de búsqueda. Es este operador el encargado de introducir nuevo material genético en la búsqueda de soluciones ya que el cruzamiento no introduce ningún material nuevo.

Cuando la codificación utilizada es binaria el operador de mutación simplemente modifica el valor de un bit en la cadena con una probabilidad de mutación dada.

2.5. Un algoritmo genético simple.

Una vez que se ha determinado la codificación, la función de adaptación, los parámetros y las variables para controlar el algoritmo y un criterio para terminar una corrida, un algoritmo genético puede ejecutarse implementando el algoritmo 2.

Algoritmo 2 Algoritmo genético simple.

Generar la población inicial $P(0)$ de forma aleatoria y hacer $t = 0$
repeat
  Evaluar el *fitness* de cada individuo en $P(t)$
  Seleccionar padres de $P(t)$ de acuerdo al *fitness*
  Efectuar cruzamiento y mutación para producir la población $P(t+1)$
  $t = t+1$
until Alcanzar el criterio de parada

2.6. Teorema de esquemas

El teorema de esquemas (*schemata*) [52] fue la primera explicación rigurosa de como funcionan los GAs. Un esquema (*schema*) es un patrón que describe un conjunto de cadenas de genes con similaridades en ciertas posiciones de la cadena y puede definirse formalmente como:

Definición 2.9. *Esquema:* Sea un espacio de representación $I = \mathcal{A}_1 \times \mathcal{A}_2, \ldots, \times \mathcal{A}_n$, para cada conjunto de alelos $\mathcal{A}_i$, se define un conjunto extendido $\mathcal{A}_i^\star = \mathcal{A}_i \cup \{\star\}$, donde $\star$ es un símbolo especial "no importa". Entonces, un esquema es cualquier miembro del conjunto $I^+ = \mathcal{A}_1^\star \times \mathcal{A}_2^\star, \ldots, \times \mathcal{A}_n^\star$.

Un esquema $H = (h_1, h_2, \ldots, h_n)$ describe un conjunto de cromosomas que tiene los mismos alelos que H en todas las posiciones i tal que $h_i \neq \star$, para una cadena binaria, esto es:

$$H = \{x \in I | \forall i \in \{1, 2, \ldots, n\} \ , \ h_i \in \{0, 1\star\}\}.$$

Las posiciones en donde H no tiene un símbolo $\star$ se llaman posiciones fijas.

Además, para poder trabajar sobre la idea de esquemas y similaridades entre distintos esquemas se definen dos métricas. El orden de un esquema, que es el número de posiciones fijas y el tamaño de definición, que es la distancia entre el primero y el último de los símbolos fijos. Las definiciones 2.10 y 2.11 describen formalmente las métricas mencionadas.

Definición 2.10. *Orden de un esquema:* El orden de un esquema $H \in I^+$, denotado por $o(H)$, es el número de posiciones fijas presentes en el esquema.

Definición 2.11. Tamaño de definición de un esquema: El tamaño de definición de un esquema $H \in I^+$, denotado por $\delta(H)$, es la distancia entre la primera y última posición de la cadena con símbolos fijos .

Como ejemplo, sea el alfabeto binario $\{0, 1\}$, agregando un símbolo especial $\star$ tenemos $\{0, 1, \star\}$, un esquema es una cadena sobre este alfabeto. El esquema $H = \star 11\star$, describe un subconjunto de S con cuatro elementos $\{0110, 0111, 1110, 1111\}$. El órden de H ($o(H)$) es 2 puesto que posee dos posiciones fijas, mientras que su distancia de definición ($\delta(H)$) es 1 ya que el primer valor fijo se encuentra en la posición 2 y el último en la 3.

El teorema de esquemas analiza el efecto de los operadores de los GAs y provee un límite inferior sobre los cambios en la tasa de muestreo para un esquema, de una generación a otra.

En un dado tiempo t existen m elementos de un esquema particular H en un población genética $P(t) = \{P_1, \ldots, P_N\}$, esto se representa por $\mathsf{m} = \mathsf{m}(H,t)$. Sea i_H la representación del índice de un individuo P_j en el conjunto $\{H \wedge P(t)\}$, $i_H \in \{1, \ldots, \mathsf{m}\}$ y f_{i_H} su correspondiente valor de adaptación. Aplicando a la población una selección por medio de ruleta ponderada, esperamos tener en el conjunto de padres un número de elementos de el esquema H igual a:

$$\mathsf{m}(H,t+1) = \frac{\mathsf{f}_{1_H}}{\sum_{j=1}^{N} P[j]_{fitness}} \cdot N + \cdots + \frac{\mathsf{f}_{\mathsf{m}_H}}{\sum_{j=1}^{N} P[j]_{fitness}} \cdot N = \frac{N}{\sum_{j=1}^{N} P[j]_{fitness}} \cdot (\mathsf{f}_{1_H} + \cdots + \mathsf{f}_{\mathsf{m}_H}) =$$

$$\frac{N}{\sum_{j=1}^{N} P[j]_{fitness}} \cdot (\sum_{i_H=1}^{m} \mathsf{f}_{i_H}) \cdot \frac{\mathsf{m}}{\mathsf{m}} = \frac{\hat{\mathsf{f}}(H)}{\hat{\mathsf{f}}(P)} \cdot \mathsf{m} = \frac{\hat{\mathsf{f}}(H)}{\hat{\mathsf{f}}(P)} \cdot \mathsf{m}(H,t) \tag{2.5}$$

donde $\hat{\mathsf{f}}(P)$ es el valor de adaptación promedio de toda la población y $\hat{\mathsf{f}}(H)$ es el valor de adaptación promedio de las cadenas que son representadas por el esquema H en $P(t)$.

La última fórmula, implica que los esquemas con valores de adaptación por encima del promedio de la población recibirán un número mayor de muestras en la siguiente generación, mientras que los esquemas con valores de adaptación por debajo del promedio de la población recibirán un número cada vez menor de muestras [50].

Ahora se considerarán los efectos del operador de cruzamiento. Existen elementos en la población que no van a ser cruzados, y estos elementos serán copiados en forma directa a la siguiente generación. Además, existe una probabilidad de ruptura de un esquema debido al operador de cruzamiento. Siendo que un esquema sobrevive al cruzamiento simple sólo cuando el punto de cruce cae fuera del tamaño de definición, entonces, la probabilidad de sobrevivencia al cruzamiento simple es $1 - \frac{\delta(H)}{l-1}$.

Por simplicidad se supondrá que el cruzamiento siempre destruye. Pero, de echo, existe una probabilidad de que elementos que no pertenezcan a H generen por cruzamiento elementos que pertenecen

a H. Sea p_c la probabilidad de cruzamiento. Entonces el número de elementos de H en $t+1$ es, al menos:

$$\mathsf{m}(H,t+1) \geq \overbrace{(1-p_c)\cdot \mathsf{m}(H,t)\cdot \frac{\hat{\mathsf{f}}(H)}{\hat{\mathsf{f}}(P)}}^{\text{no elegidos para cruzamiento}} + \underbrace{(p_c)\cdot \mathsf{m}(H,t)\cdot \frac{\hat{\mathsf{f}}(H)}{\hat{\mathsf{f}}(P)}\cdot \overbrace{(1-\frac{\delta(H)}{l-1})}^{\text{no destruidos}}}_{\text{elegidos para cruzamiento}}$$

$$\mathsf{m}(H,t+1) \geq \mathsf{m}(H,t)\cdot \frac{\hat{\mathsf{f}}(H)}{\hat{\mathsf{f}}(P)}\cdot [1-p_c+p_c-p_c\cdot \frac{\delta(H)}{l-1}] = \mathsf{m}(H,t)\cdot \frac{\hat{\mathsf{f}}(H)}{\hat{\mathsf{f}}(P)}\cdot [1-p_c\cdot \frac{\delta(H)}{l-1}] \qquad (2.6)$$

Entonces, considerando la selección y el cruzamiento a la vez, aquellos esquemas con valor de adaptación por encima del promedio y tamaño de definición pequeño reciben más muestras para la siguiente generación.

El último operador a considerar es la mutación de un bit. Para que un esquema sobreviva, la mutación no debe ocurrir en una posición con un valor fijo. Siendo p_m la tasa de mutación, la posibilidad de que un valor de un bit permanezca es $(1-p_m)$, entonces la probabilidad de sobrevivir para un esquema es $(1-p_m)^{o(H)}$, para pequeños valores de p_m ($p_m \ll 1$) la probabilidad de supervivencia del esquema puede ser aproximada por $1-o(H)\cdot p_m$. Entonces considerando todos los operadores la expresión para $\mathsf{m}(H,t+1)$ es:

$$\mathsf{m}(H,t+1) \geq \mathsf{m}(H,t)\cdot \frac{\hat{\mathsf{f}}(H)}{\hat{\mathsf{f}}(P)}\cdot [1-p_c\cdot \frac{\delta(H)}{l-1} - o(H)\cdot p_m] \qquad (2.7)$$

En conclusión, los esquemas cortos, de orden bajo, con adaptación por encima del promedio reciben muestras de manera exponencialmente creciente en cada generación. Esto se conoce como teorema de esquemas [50, 52].

Muchos esquemas diferentes son muestreados cuando una población de cadenas es evaluada, y de echo, son muestreados más esquemas que el número de cadenas contenidas en la población. El efecto acumulativo de evaluar una población de puntos provee información estadística acerca de cualquier

subconjunto particular de esquemas. Esto implica que muchas competencias entre esquemas son resueltas en forma paralela, Holland llamó a esto *paralelismo implícito* [52].

Es importante recalcar que el término paralelismo implícito no se refiere a la potencialidad de implementar algoritmos genéticos paralelos. De echo, Holland primero uso el término paralelismo intrínseco en [52], pero luego decidió cambiarlo al término paralelismo implícito para evitar confusiones con la terminología de la computación paralela. Desafortunadamente, el término paralelismo implícito en la comunidad de cómputo paralelo se refiere a paralelismo que es obtenido a partir de código escrito en lenguajes funcionales que no tienen constructores paralelos explícitos [93].

2.7. Convergencia y diversidad

Luego de varias generaciones, es posible que el operador de selección conduzca a todos los bits en algunas posiciones a un sólo valor (0 o 1 si se utiliza un código binario). Un gen se dice que ha convergido cuando al menos el 95 % de la población comparte el mismo valor. Una población se dice que convergió cuando todos los genes han convergido [27].

Una vez que la población converge, la habilidad de un GA para continuar buscando mejores soluciones por cruzamiento desaparece. Cuando los individuos padres poseen cromosomas idénticos, el intercambio no produce ningún efecto y los individuos generados tienen cromosomas iguales al de sus padres. Sólo el operador de mutación permanece para explorar nuevas soluciones. Cuando esto ocurre, la búsqueda llevada a cabo por el algoritmo genético es sólo una búsqueda aleatoria. Idealmente, la convergencia debería ocurrir sólo cuando la solución es alcanzada. Si la convergencia ocurre sin que el GA alcance una solución satisfactoria, se dice que el GA ha convergido en forma prematura.

La preponderancia de individuos con un valor de adaptación demasiado elevado con respecto al promedio de la población lleva a que los métodos de selección estándar converjan en forma pre-

matura. Este es un problema particularmente importante cuando se trabaja con poblaciones genéticas pequeñas. Otro problema ocurre cuando todos los individuos tienen probabilidad de selección similar y el GA posee una convergencia demasiado lenta. La convergencia hacia un pico u otro sin una ventaja importante entre un punto y otro es causado por la similaridad genética [50].

Para prevenir problemas causados por la similaridad genética, se han desarrollado varios métodos. Los métodos de nicho (*niching*), por ejemplo, son técnicas que fomentan la formación y mantenimiento de subpoblaciones estables en GAs. El mantenimiento de la diversidad permite a los algoritmos genéticos buscar muchos picos en paralelo, evitando quedar atrapados en óptimos locales del espacio de búsqueda. Los métodos de *niching* pueden ser aplicados para la formación y el mantenimiento de subsoluciones interinas en la búsqueda de una única solución final. Sin embargo, estas técnicas han sido tradicionalmente utilizadas en el contexto de la formación y el mantenimiento de múltiples soluciones finales [62, 63]. El método de repartija de *fitness* (*fitness sharing*) es probablemente el mejor y más conocido método de entre los métodos de *niching* [73].

2.7.1. Fitness Sharing

Los métodos de *fitness sharing* fueron introducidos en 1987 por Goldberg y Richardson para modificar el horizonte de búsqueda reduciendo el valor de adaptabilidad de los individuos que poseen cierta similaridad con respecto a otros individuos de una población [62]. Los métodos de *fitness sharing* requieren que el valor de adaptación sea compartido entre individuos similares como un sólo recurso. Para esto a cada individuo $P[i]$ se le asigna un valor llamado su *cuenta de nicho*, denotado por $P[i]_{ncount}$, que se utiliza como indicador del nivel de similaridad del individuo. El *fitness compartido* de un individuo $P[i]$ es igual a su viejo valor de adaptación, dividido este valor.

El valor de la cuenta de nicho para un individuo se calcula a partir de una función *sharing*. Ésta mide el nivel de similaridad entre dos elementos de la población. La función retorna 1 si los elementos

son idénticos, 0 si cruzan algún umbral de diferenciabilidad, y un nivel intermedio en un nivel intermedio de diferenciabilidad. El umbral de diferenciabilidad es especificado por una constante, σ_{share}, el cual indica la máxima distancia permitida para que un *sharing* diferente de cero ocurra. Típicamente la función de *sharing* es:

$$sh(d(P_i,P_j)) = \begin{cases} 1-(\frac{d(P_i,P_j)}{\sigma_{share}})^{\alpha}, & \text{si } d(P_i,P_j) \leq \sigma_{share} \\ 0, & \text{de otro modo} \end{cases} \tag{2.8}$$

donde :

α es un parámetro constante el cual regula la forma de la función de *sharing*, y

d es una función de distancia dada, que calcula la distancia entre sus argumentos.

Cuando se implementa en el espacio objetivo, la función d usualmente corresponde a la distancia Euclidiana [35]. Mientras que, cuando se implementa en el espacio de genes, es usual utilizar el número de diferencias de bits (o distancia de Hamming) entre las soluciones. Tal medida de la distancia es útil en los problemas definidos en el espacio de los genes [35].

El valor de la *cuenta del nicho* para $P[i]$ es igual a la sumatoria de los valores de la función de *sharing* entre $P[i]$ y cada individuo de la población $P[j]$

$$P[i]_{ncount} = \sum_{j=1}^{N} sh(d(P_i,P_j)) \tag{2.9}$$

Finalmente, el valor de adaptación compartido del elemento $P[i]$ es:

$$P[i]_{fitness} = \frac{P[i]_{fitness}}{P[i]_{ncount}} \tag{2.10}$$

2.7.2. Restricción de apareamiento

Cuando el procedimiento de *niching* se implementa en problemas con objetivo único, Deb y Goldberg [35] notaron que la recombinación entre cromosomas de nichos diferentes usualmente producen individuos pobremente adaptados. Esto lleva a la degradación del desempeño del GA. Con el fin de evitar este problema, en [35] se propone restringir el apareamiento (*matting restriction*) sólo entre individuos que están a una cierta distancia en el espacio genotípico.

2.7.3. Elitismo

Un algoritmo evolutivo opera a través de la utilización iterativa de operadores genéticos. Estos operadores genéticos pueden producir la pérdida de una buena solución encontrada en una generación, en generaciones posteriores. En vista a esto, en [27] se sugiere una política en la cual siempre se incluye el mejor individuo de $P(t)$ en $P(t+1)$ a fin de preservarlo de la posible acción negativa de los operadores genéticos. A esta estrategia se le conoce como elitismo. Se ha demostrado formalmente que el elitismo asegura la convergencia de los métodos evolutivos [70].

Capítulo 3

Optimización Multiobjetivo

En el presente capítulo se describen los principios de optimización multiobjetivo formalizando los conceptos principales. Así mismo, se describe el modelo general del proceso de solución utilizado con problemas de optimización multiobjetivo. Además, se presentan algunos métodos clásicos representativos utilizados para la resolución de este tipo de problemas, discutiendo sus ventajas y desventajas potenciales.

3.1. Conceptos básicos y terminología

La mayoría de los problemas del mundo real requieren la búsqueda de soluciones que optimicen de manera simultánea varios objetivos. Esto es así no sólo en cuestiones relacionadas con la ingeniería o la toma de decisiones para los negocios, sino más bien, forma parte de la vida diaria de las personas. Por ejemplo, cuando las madres seleccionan una receta de cocina ellas desean aquella que, ajustándose a un determinado presupuesto, proporcione a sus hijos la mejor y más adecuada carga de vitaminas, proteínas y minerales de acuerdo con la edad y las necesidades alimentarías de estos. Así mismo, desean que la comida a preparar se ajuste a los gustos, posiblemente diversos, de los distintos

comensales. Esta situación sencilla de la vida diaria permite señalar algunas de las características más importantes de los problemas de optimización con objetivos múltiples:

- los distintos objetivos pueden ser conflictivos, tener distintas unidades de medidas, y ser inconmensurables,
- al existir objetivos contradictorios no existe una solución única que sea mejor que otras con respecto a todos los objetivos, sino que se tiene un conjunto de alternativas de solución, las cuales representan los mejores compromisos entre los distintos objetivos,
- de las alternativas existentes sólo interesa el conjunto de soluciones que cumplen con ciertas restricciones (soluciones factibles) y finalmente,
- existiendo un conjunto de alternativas posibles es necesario que un tomador de decisiones proporcione información de preferencia a fin de seleccionar la solución a ser implementada.

En los problemas de optimización con objetivo único, el espacio de búsqueda se encuentra totalmente ordenado por medio de la función objetivo por lo que es posible hablar de *un* óptimo global el cual se define formalmente como [86]:

Definición 3.1. *Óptimo global (monobjetivo):* dada una función $f : \mathbb{R}^n \rightarrow \mathbb{R}$, para $\mathbf{x} \in \mathbb{R}^n$ el valor $f^\star = f(\mathbf{x}^\star)$ es llamado un óptimo global (máximo o mínimo) sí y sólo sí:

$$\forall \mathbf{x} \in \mathbb{R}^n : f(\mathbf{x}^\star) \textit{ es mejor que } f(\mathbf{x}) \tag{3.1}$$

Entonces, $\mathbf{x}^\star$ es la solución global óptima, f es la función objetivo. El problema de determinar la solución global óptima es llamado *problema de optimización global*.

Así, cuando se tiene un objetivo único, el proceso de optimización se reduce a hallar el máximo o el mínimo valor de una función. La matemática cuenta con herramientas adecuadas para esto por lo que

no resulta extraño que el enfoque usual de los métodos tradicionales para la resolución de problemas donde se contemplan objetivos múltiples sea el combinar de algún modo los distintos objetivos en una única función para luego resolver el problema monobjetivo. A estos métodos se les conoce como métodos de escalarización puesto que transforman una función de optimización vectorial $\mathbf{F} \in \mathbb{R}^n \rightarrow \mathbb{R}^k$ en una función de optimización escalar $\mathbf{F}' = f \in \mathbb{R}^n \rightarrow \mathbb{R}$.

Los métodos de escalarización han demostrado ser adecuados para la resolución de un gran número de problemas donde se consideran objetivos múltiples. Sin embargo, cuando se consideran simultáneamente objetivos contradictorios, con distintas unidades de medida o inconmensurables, generalmente resulta inadecuado combinar los diferentes objetivos en una única medida de desempeño. Así mismo, para la gran mayoría de los problemas, no resulta posible priorizar un único objetivo, sino más bien todos deben ser considerados en forma simultánea. En estos casos, se tiene un problema de optimización multiobjetivo (*Multiobjective Optimization Problem* - MOP).

Mientras que la optimización con objetivo único busca un vector de decisión *n*-dimensional que optimice una función escalar, en optimización multiobjetivo se intenta encontrar uno que optimice una función vectorial cuyos elementos representan las distintas funciones objetivo. Formalmente un problema de optimización multiobjetivo puede definirse como:

Definición 3.2. *Problema de Optimización Multi-Objetivo:* un MOP general optimiza una función

$$\mathbf{y} = \mathbf{F}(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_k(\mathbf{x})) \tag{3.2}$$

sujeto a

$$\mathbf{g}(\mathbf{x}) = (g_1(\mathbf{x}), \ldots, g_m(\mathbf{x})) \leq \mathbf{0} \tag{3.3}$$

donde

$$\mathbf{x} = (x_1, \ldots, x_n) \in \mathcal{X} \subseteq \mathbb{R}^n$$

$$\mathbf{y} = (y_1, \ldots, y_k) \in \mathcal{Y} \subseteq \mathbb{R}^k$$

x es una variable de decisión vectorial *n*-dimensional, **y** es un vector objetivo *k*-dimensional, $\mathcal{X} \subseteq \mathbb{R}^n$ denota el espacio de decisión, e $\mathcal{Y} \subseteq \mathbb{R}^k$ denota el espacio objetivo.

El conjunto de restricciones dadas por 3.3 define la región de factibilidad $\mathcal{X}_f \subseteq \mathbb{R}^n$ y cualquier punto $\mathbf{x} \in \mathcal{X}_f$ es una solución factible.

Definición 3.3. *Conjunto factible:* el conjunto factible $\mathcal{X}_f$ está definido como el conjunto de vectores de decisión **x** que satisfacen las restricciones $\mathbf{g}(\mathbf{x})$:

$$\mathcal{X}_f \;=\; \{\, \mathbf{x} \in \mathcal{X} \mid \mathbf{g}(\mathbf{x}) \leq \mathbf{0} \,\} \tag{3.4}$$

La imagen de $\mathcal{X}_f$, es la región de factibilidad en el espacio objetivo y se denota por

$$\mathcal{Y}_f = \mathbf{F}(\mathcal{X}_f) = \bigcup_{\mathbf{x} \in \mathcal{X}_f} \{\mathbf{F}(\mathbf{x})\} \tag{3.5}$$

Como se dijo anteriormente, en optimización con un solo objetivo, el espacio de búsqueda (el conjunto de variables de decisión factibles) está completamente ordenado de acuerdo al valor de la función objetivo. Esto es, dadas dos soluciones **a** y **b** en el espacio de soluciones factibles, se cumple sólo una de las siguientes proposiciones: $f(\mathbf{a}) > f(\mathbf{b})$, $f(\mathbf{a}) = f(\mathbf{b})$ o $f(\mathbf{a}) < f(\mathbf{b})$. Sin embargo, cuando se busca la solución de un MOP con objetivos contradictorios, el espacio de búsqueda se encuentra sólo parcialmente ordenado y dos soluciones pueden ser indiferentes entre sí siendo poco usual que una única variable de decisión optimice de manera simultánea todas las funciones objetivo. Por tanto, no existe una solución única que pueda ser considerada como "la mejor", sino un conjunto de soluciones que representan los mejores compromisos entre los distintos objetivos.

Considere, por ejemplo, el diseño de una red de transmisión de datos. Teniendo en cuenta sólo el costo y la confiabilidad, en función del número de paquetes perdidos o alterados como una fracción del total de enviados, un diseño óptimo sería uno que minimice los costos y la tasa de error residual. Si tal

solución existe, se tiene un problema de optimización de objetivo único pues la solución óptima para cualquier objetivo es también la solución óptima para la otra. Sin embargo, en el caso del ejemplo, los objetivos son usualmente conflictivos: la red más económica por lo general es de menor confiabilidad, y una red con mayor confiabilidad por lo general es más costosa. Puesto que se tienen objetivos contradictorios, éstos no pueden ser optimizados simultáneamente y no existe una solución única sino, en vez de ello, se tiene un conjunto de soluciones satisfactorias que representen un compromiso entre los dos objetivos considerados. Dependiendo de las consideraciones del diseñador, una solución intermedia puede resultar aceptable.

El conjunto de soluciones de compromiso es un conjunto de soluciones óptimas en el sentido que cada una es mejor que las otras en algún objetivo, pero ninguna puede considerarse mejor cuando se tienen en cuenta todos los objetivos de forma simultánea. La existencia de múltiples soluciones óptimas hace necesaria una noción distinta de óptimo .

La noción de óptimo para problemas multiobjetivo más comúnmente aceptada es una propuesta original de Francis Ysidro Edgeworth, y generalizada posteriormente por Vilfredo Pareto, llamada óptimo de Pareto (*Pareto Optimum*) [23]. Los conceptos esenciales para entender la noción de óptimo de Pareto son: dominancia de Pareto (*Pareto Dominance*), optimalidad de Pareto (*Pareto Optimality*), conjunto Pareto óptimo (*Pareto Optimal Set*), y frente de Pareto (*Pareto Front*) [23], definidos a continuación.

Definición 3.4. *Dominancia de Pareto:* un vector $\mathbf{u} = (u_1, \ldots, u_k)$ se dice que domina a otro vector $\mathbf{v} = (v_1, \ldots, v_k)$ sí y sólo sí: **u** es parcialmente mejor que **v**, esto es

$$\forall i \in \{1, \ldots, k\}, u_i \text{ es mejor o igual que } v_i \ \wedge \ \exists i \in \{1, \ldots, k\} | u_i \text{ es estrictamente mejor que } v_i$$

Es decir, **u** domina a **v** si **u** es mejor o igual a **v** en todos los objetivos y estrictamente mejor en al menos un objetivo. En este caso, se dice que existe una dominancia de **u** respecto a **v** lo que se denota

como $\mathbf{u} \succ \mathbf{v}$ (**u** es mejor que **v**).

Note que la definición anterior de dominancia Pareto es independiente de que se trate de minimizaciones, maximizaciones o combinaciones de estas.

Definición 3.5. *Dominancia de Pareto débil:* un vector $\mathbf{u} = (u_1, \ldots, u_k)$ se dice que domina de forma débil a otro vector $\mathbf{v} = (v_1, \ldots, v_k)$ sí y sólo sí: **u** es parcialmente mejor o igual que **v**, esto es

$$\forall i \in \{1, \ldots, k\}, u_i \text{ es mejor o igual que } v_i$$

Es decir $\mathbf{u} \succ \mathbf{v}$ o $\mathbf{u} = \mathbf{v}$. La dominancia débil de **u** respecto a **v** se denota como $\mathbf{u} \succeq \mathbf{v}$.

Definición 3.6. *Dominancia de Pareto Estricta:* un vector $\mathbf{u} = (u_1, \ldots, u_k)$ se dice que domina de forma estricta a otro vector $\mathbf{v} = (v_1, \ldots, v_k)$ sí y sólo sí: **u** es mejor que **v** para todos los objetivos considerados, esto es

$$\forall i \in \{1, \ldots, k\}, u_i \text{ es mejor } v_i$$

La dominancia estricta de **u** respecto a **v** se denota como $\mathbf{u} \succ\succ \mathbf{v}$.

Definición 3.7. *Relación de Nodominancia:* un vector $\mathbf{u} = (u_1, \ldots, u_k)$ se dice que nodomina al vector $\mathbf{v} = (v_1, \ldots, v_k)$ sí y sólo si la relación $\mathbf{u} \succ \mathbf{v}$ no se cumple. La relación de *nodominancia* del vector **u** respecto al vector **v** se denota como $\mathbf{u} \not\succ \mathbf{v}$ y se lee **u** *nodomina* a **v**.

Definición 3.8. *Vectores no comparables:* los vectores **u** y **v** son no comparables o indiferentes sí y sólo si:

$$(\mathbf{u} \not\succ \mathbf{v}) \wedge (\mathbf{v} \not\succ \mathbf{u}) \wedge (\mathbf{v} \neq \mathbf{u})$$

En este caso, ni **u** es mejor que **v**, ni **v** puede ser considerado mejor que **u**. Si bien los vectores **u** y **v** no son estrictamente iguales, ambas soluciones pueden ser consideradas igualmente buenas debido a que ninguna es superior a la otra cuando consideramos todos los objetivos. Se dice entonces que el vector **v** es indiferente o no comparable al vector **u** y se lo denota como $\mathbf{v} \sim \mathbf{u}$.

Relación	*Ejemplo*	*Significado*
Dominancia estricta	$\mathbf{u} \succ\succ \mathbf{v}$	**u** es mejor que **v** en todos los objetivos
Dominancia	$\mathbf{u} \succ \mathbf{v}$	**u** domina a **v**
Nodominancia	$\mathbf{u} \not\succ \mathbf{v}$	**u** nodomina a **v**
Dominancia débil	$\mathbf{u} \succeq \mathbf{v}$	**u** no es peor que **v** en ningún objetivo
No comparable	$\mathbf{u} \sim \mathbf{v}$	ni **u** domina débilmente a **v**, ni **v** domina débilmente a **u**

Tabla 3.1. Notación utilizada para las relaciones de dominancia Pareto entre vectores objetivo.

La Tabla 3.1 resume la notación utilizada en el presente trabajo para los conceptos de dominancia Pareto. Definidas las distintas relaciones de dominancia, se define formalmente un óptimo de Pareto y el conjunto Pareto Óptimo como:

Definición 3.9. *Óptimo de Pareto:* se dice que una solución $\mathbf{x} \in \mathcal{X}_f$ es Pareto óptima con respecto a un conjunto $\Omega \subseteq \mathcal{X}_f$ si y sólo si:

$\nexists \mathbf{x}' \in \Omega$ para el cual $\mathbf{v}' = \mathbf{F}(\mathbf{x}') = (f_1(\mathbf{x}'), \ldots, f_k(\mathbf{x}'))$ domina a $\mathbf{u} = \mathbf{F}(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_k(\mathbf{x}))$

Si en el contexto resulta claro a que conjunto Ω se hace referencia, simplemente no se lo menciona. La solución **x** es una solución Pareto óptima si y solo si es no dominada con respecto al conjunto $\mathcal{X}_f$.

Definición 3.10. *Conjunto Pareto Óptimo:* para un MOP, el conjunto Pareto óptimo, denotado por $\mathcal{P}^\star$, se define como:

$$\mathcal{P}^\star = \{\mathbf{x} \in \mathcal{X}_f | \ \nexists \mathbf{x}' \in \mathcal{X}_f \text{ para el cual } \mathbf{F}(\mathbf{x}') \text{ domine a } \mathbf{F}(\mathbf{x})\}$$

En palabras, un vector **x** es Pareto óptimo si no existe ningún otro vector de decisión factible $\mathbf{x}' \in \mathcal{X}_f$ que lo domine. El conjunto de soluciones en un MOP, denotado por $\mathcal{P}^\star$ está compuesto por todos estos vectores Pareto óptimos. Las soluciones Pareto óptimas son también llamadas soluciones no inferiores, admisibles, o soluciones eficientes, mientras que sus vectores objetivo correspondientes son denominados *no dominados*. El conjunto de vectores no dominados en el espacio objetivo, correspondiente al conjunto de soluciones $\mathcal{P}^\star$, es llamado Frente Pareto Optimo y denotado por $\mathcal{PF}^\star$ [86]. Formalmente, el Frente Pareto Optimo puede definirse como:

Definición 3.11. *Frente Pareto Óptimo:* dado un problema de optimización mutiobjetivo $\mathbf{F}(\mathbf{x})$ y un conjunto Pareto Óptimo $\mathcal{P}^\star$, el Frente Pareto Óptimo $\mathcal{PF}^\star$ se define como:

$$\mathcal{PF}^\star = \{\mathbf{u} = \mathbf{F}(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_k(\mathbf{x})) | \, \mathbf{x} \in \mathcal{P}^\star\}$$

Con el fin de ilustrar los distintos conceptos presentados, considere el problema de optimización multiobjetivo conocido como el problema de prueba $\mathbf{F}_2$ de Schaffer [76]:

$$\begin{aligned} \textit{Minimizar} \quad & \mathbf{F}_2(x) = (f_1(x), f_2(x)) \\ \textit{donde}: \quad & f_1(x) = x^2, \\ & f_2(x) = (x-2)^2, \\ & x \in [-5; 5] \end{aligned} \tag{3.6}$$

La Figura 3.1 presenta la gráfica de las dos funciones que conforman la función multiobjetivo considerada en la Ecuación 3.6 en el rango $[-5; 5]$. Note que la función f_1 alcanza el valor mínimo cuando x se hace cero, mientras que el mínimo para f_2 se obtiene cuando x es igual a dos. Para valores inferiores a cero, ambas funciones tienen pendientes de signo igual. Cuando x se hace cero, la pendiente de f_1 cambia de signo, mientras que f_2 cambia de signo recién en $x = 2$. Entonces, para los valores de x en $[0; 2]$, mientras que en una función se disminuye, en otra se aumenta. Por tanto, el conjunto Pareto Óptimo para el MOP planteado en 3.6 es:

$$\mathcal{P}^\star = \{x \mid 0 \leq x \leq 2\}$$

Para cualquier solución que no se encuentre en el conjunto $\mathcal{P}^\star$, existe siempre una solución que la domina.

El Frente Pareto Óptimo correspondiente está formado por el conjunto de vectores bidimensionales obtenidos al evaluar la función $\mathbf{F}_2$ para cada valor de $x \in \mathcal{P}^\star$. La Figura 3.2 representa los valores de la función $f_1(x)$ graficados contra la función $f_2(x)$ para $x \in [0; 2]$. Es decir, es la gráfica de los elementos en $\mathcal{P}F^\star$ para el problema 3.6.

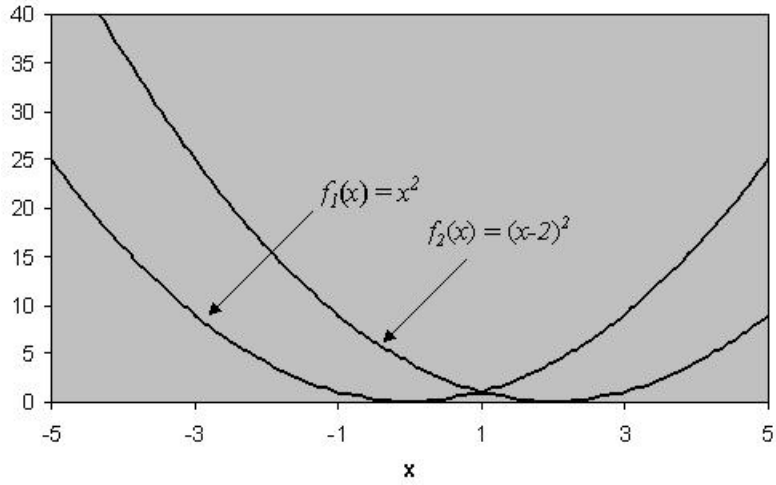

Figura 3.1. Valores de $f_1 = x^2$ y $f_2 = (x-2)^2$ en función de $x \in [-5;5]$.

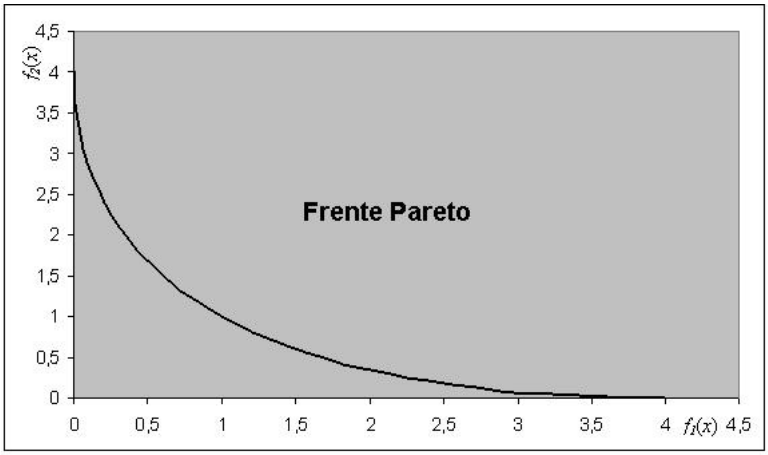

Figura 3.2. Frente Pareto Óptimo para el problema $\mathbf{F}_2$ de Schaffer.

3.2. Proceso de solución de un MOP

Un problema de optimización multiobjetivo se considera matemáticamente resuelto cuando el conjunto Pareto Óptimo es encontrado. En problemas reales, usualmente se requiere de una única solución. Puesto que desde un punto de vista estrictamente matemático todas las soluciones en el conjunto Pare-

to son igualmente buenas, para obtener *la* solución a ser implementada en un caso concreto, se necesita información adicional de preferencia. Un tomador de decisiones (*decision maker* - DM) humano es quien se encarga de proporcionar la información de preferencia necesaria para seleccionar una solución del conjunto Pareto. Así, el proceso de solución de un MOP puede dividirse en dos procesos conceptualmente distintos [53, 64]:

- *Proceso de búsqueda u optimización,* por el cual se explora el conjunto de soluciones factibles en busca de soluciones Pareto Optimas.

- *Proceso de toma de decisiones,* por el cual se selecciona una solución de compromiso adecuada, a partir del Conjunto Pareto Óptimo hallado por el proceso anterior.

La importancia de la toma de decisiones como parte del proceso de solución es una buena razón para clasificar los distintos métodos para solucionar problemas de optimización multiobjetivo de acuerdo con la manera en que se combinan la búsqueda del conjunto de soluciones y la toma de decisiones [64]. Así, los métodos se clasifican en:

- **Métodos a priori:** la toma de decisiones se realiza antes de la búsqueda de soluciones. Usualmente los distintos objetivos se combinan en uno, el cual implícitamente incluye información de preferencia proporcionada por el tomador de decisiones. Esto hace que efectivamente el MOP se convierta en un problema de optimización monobjetivo, antes de la optimización.

- **Métodos a posteriori:** la toma de decisiones se realiza de forma posterior a la búsqueda. La optimización se realiza sin ninguna información de preferencia. El resultado del proceso de búsqueda es un conjunto de soluciones candidatas, idealmente Pareto-Optimas, a partir de la cual la decisión final será realizada por el tomador de decisiones, conociendo las alternativas disponibles.

- **Métodos interactivos o progresivos:** la toma de decisiones se realiza durante la búsqueda de soluciones de manera interactiva. Luego de cada paso de optimización, un conjunto de soluciones de compromiso es presentado al tomador de decisiones, quien proporcionará información de preferencia que guiará el proceso de búsqueda.

Con los métodos a priori, donde se combinan múltiples objetivos en un único criterio de optimización, cuando son aplicables, se tiene la ventaja que las estrategias clásicas de optimización monobjetivo pueden ser utilizadas sin modificaciones. La desventaja es que requiere un conocimiento profundo del dominio del problema que permita realizar la escalarización de forma correcta. Este conocimiento del dominio del problema requerido por los métodos a priori usualmente no se encuentra disponible. Es más, en varios problemas de diseño en ingeniería, lo que se desea específicamente es ganar mayor conocimiento sobre el problema y sobre las soluciones alternativas. Realizar la toma de decisiones luego del proceso de búsqueda (métodos a posteriori) resuelve el problema, pero excluye la articulación de la preferencia del tomador de decisiones, lo que podría a su vez reducir la complejidad del espacio de búsqueda. Los métodos interactivos superan las desventajas de los métodos a priori y posteriori permitiendo al tomador de decisiones expresar sus preferencias a medida que avanza en el conocimiento del problema considerado.

Como se dijo anteriormente, los métodos tradicionales de búsqueda de óptimos para problemas multiobjetivo se basan en convertir el problema considerado en un problema monobjetivo. Estos métodos encuentran una solución a la vez por lo que se han propuesto varios métodos para aproximar un conjunto de soluciones en vez de un único punto. En la siguiente sección se presentan algunos enfoques tradicionales, representativos para la solución de MOPs, seguida de una discusión sobre sus principales características. En el próximo capítulo se introducen los algoritmos evolutivos como una técnica actual para la resolución de estos problemas.

3.3. Métodos tradicionales de optimización

3.3.1. Métodos de suma ponderada

En los métodos de suma ponderada las funciones objetivo del problema de optimización multiobjetivo original se combinan de forma lineal utilizando diferentes coeficientes de peso para formar una función escalar a ser optimizada. Esto significa que un problema original de optimización con objetivos múltiples:

$$Optimizar \qquad \mathbf{F}(\mathbf{x}) = (f_1(\mathbf{x}, \ldots, f_k(\mathbf{x})) \tag{3.7}$$

sujeto a:

$$\mathbf{g}(\mathbf{x}) = (g_1(\mathbf{x}), \ldots, g_m(\mathbf{x})) \leq \mathbf{0} \tag{3.8}$$

se transforma en un problema de optimización de la forma:

$$Optimizar \qquad y = f(\mathbf{x}) = w_1 \cdot f_1(\mathbf{x}) + \cdots + w_k \cdot f_k(\mathbf{x}) = \sum_{i=1}^{k} w_i \cdot f_i(\mathbf{x}) \tag{3.9}$$

sujeto a

$$\mathbf{g}(\mathbf{x}) = (g_1(\mathbf{x}), \ldots, g_m(\mathbf{x})) \leq \mathbf{0} \tag{3.10}$$

donde

$$w_i \geq 0 \text{ para todo } i = 1, \ldots, k, \sum_{i=1}^{k} w_i = 1 \tag{3.11}$$

Los métodos de suma ponderada pueden ser utilizados tanto a priori, de forma interactiva o a posteriori. Como método a priori, el tomador de decisiones expresa su preferencia por los diferentes objetivos seleccionando un vector de pesos dado y evaluando la función $f(\mathbf{x})$ correspondiente para obtener *una* solución. El problema es que el tomador de decisiones no puede estar seguro de reflejar la importancia de cada objetivo original de forma apropiada. La única solución obtenida por el método de suma ponderada utilizando un vector de pesos seleccionado a priori no brinda información al tomador

de decisiones sobre otras alternativas de solución y debe confiar en su decisión inicial. La obtención de un conjunto de soluciones alternativas sería de indudable utilidad.

Para obtener un conjunto de soluciones utilizando el método de suma ponderada, es necesario resolver la Ecuación 3.9 utilizando diferentes combinaciones de pesos. Así, el proceso de búsqueda de soluciones se inicia con la selección de un vector inicial de pesos $\mathbf{w}^0$ con el que se obtiene una solución. Esta selección puede hacerse bien sea de acuerdo a las preferencias del tomador de decisiones o en forma aleatoria. Luego, el vector de pesos inicial se modifica para obtener uno nuevo. El proceso se repite hasta que un número dado de soluciones sea encontrado.

En [14] se proponen varias formas de perturbar los pesos para obtener diferentes soluciones Pareto óptimas. Así, los pesos no reflejan proporcionalmente la importancia relativa de los objetivos, sino que son sólo factores que, al variarse, localizan puntos diferentes en el conjunto Pareto. Luego que un conjunto de soluciones ha sido obtenido, las soluciones se presentan al tomador de decisiones para que se seleccione una que le resulte satisfactoria. La principal desventaja con este método es que no se pueden generar todas las soluciones Pareto Optimas cuando el espacio de soluciones compromiso no es convexo. Además, los métodos numéricos que pueden ser utilizados para buscar el óptimo de la función 3.9 la localización de puntos depende no sólo de w_i, sino también de las unidades en las que se expresen las funciones.

Así mismo, el método puede utilizarse de forma interactiva donde una vez seleccionada un vector de pesos inicial el tomador de decisiones interfiere ajustando este vector expresando su preferencia a medida que el proceso de búsqueda de soluciones se realiza, lo que implica una pérdida de tiempo para el tomador de decisiones.

3.3.2. Programación de metas

En este método, el tomador de decisiones tiene que asignar objetivos o metas que desee alcanzar para cada objetivo. Estos valores se incorporan en el problema como restricciones adicionales. La técnica tratará entonces de minimizar las desviaciones absolutas de cada objetivo con respecto a lo deseado. La forma más simple de este método puede formularse de la manera siguiente:

$$Minimizar \quad \sum_{i=1}^{k} |\frac{f_i(\mathbf{x}) - T_i}{T_i}| \text{ sujeta a } \mathbf{x} \in \mathcal{X}_f \tag{3.12}$$

donde T_i denota la meta u objetivo establecido por el tomador de decisiones para la i-ésima función objetivo f_i. El criterio es, entonces, minimizar la suma de los valores absolutos de las diferencias entre los valores deseados y los obtenidos.

Esta técnica puede ser muy eficiente en términos de tiempo de procesamiento si conocemos las metas que deseamos obtener y si éstas se encuentran en la zona factible, aunque para ello, el tomador de decisiones tiene la tarea de encontrar pesos o prioridades adecuadas para los objetivos lo cual es difícil para la mayoría de los casos. Así mismo, si la región factible es difícil de localizar, este método puede volverse muy ineficiente. Sin embargo, la técnica puede ser muy útil en casos en los que se pueden efectuar aproximaciones lineales parciales de las funciones objetivo, debido a la disponibilidad de excelentes programas para esa tarea y a la posibilidad de eliminar las metas dominadas fácilmente. Por otra parte, en los casos no lineales, otras técnicas resultarán generalmente más eficientes.

3.3.3. Ordenamiento Lexicográfico

En el ordenamiento lexicográfico, las funciones objetivo se ordenan tomando en cuenta la importancia de cada objetivo antes de iniciar el proceso de búsqueda de soluciones. Luego la solución óptima se obtiene minimizando las funciones objetivo, empezando con la más importante y procediendo de acuerdo con el orden de importancia asignado a cada uno de los objetivos. Supongamos que los subíndices de

los objetivos indican no sólo el número de función objetivo al que corresponden, sino también la prioridad de cada objetivo. Tendremos entonces que $f_1(\mathbf{x})$ y $f_k(\mathbf{x})$ denotan las funciones objetivo más y menos importantes respectivamente. Entonces, el primer problema se formula como

$$Optimizar \qquad f_1(\mathbf{x}) \tag{3.13}$$

sujeta a:

$$\mathbf{g}(\mathbf{x}) = (g_1(\mathbf{x}), \ldots, g_m(\mathbf{x})) \leq \mathbf{0} \tag{3.14}$$

obteniendo su solución $\mathbf{x_1^\star}$ y $f_1^\star = f_1(\mathbf{x_1^\star})$. Posteriormente, el segundo problema se formula como

$$Optimizar \qquad f_2(\mathbf{x}) \tag{3.15}$$

sujeta a:

$$\begin{aligned} \mathbf{g}(\mathbf{x}) &= (g_1(\mathbf{x}), \ldots, g_m(\mathbf{x})) \leq \mathbf{0} \\ f_1(\mathbf{x}) &= f_1^\star \end{aligned} \tag{3.16}$$

y la solución a este problema se obtiene como $\mathbf{x}_2^\star$ y $f_2^\star = f_2(\mathbf{x}_2^\star)$. Este procedimiento se repite hasta que todos los k objetivos han sido considerados. El k-ésimo problema está dado por:

$$Optimizar \qquad f_k(\mathbf{x}) \tag{3.17}$$

sujeta a

$$\begin{aligned} \mathbf{g}(\mathbf{x}) &= (g_1(\mathbf{x}), \ldots, g_m(\mathbf{x})) \leq \mathbf{0} \\ f_1(\mathbf{x}) &= f_1^\star, \\ &\vdots \\ f_{k-1}(\mathbf{x}) &= f_{k-1}^\star. \end{aligned} \tag{3.18}$$

La solución obtenida al final, es decir $\mathbf{x}_k^\star$, se considera como la solución deseada del problema. Este método es adecuado sólo cuando la importancia de cada objetivo es conocida y puede determinarse claramente. Esto en la práctica no es lo usual.

3.3.4. El método de las restricciones ε

Este método se basa en la optimización de una función objetivo (la principal o preferida) y considera a los demás objetivos como restricciones que están acotadas por ciertos niveles permisibles ε_i. Por lo tanto, se efectúa una optimización con un sólo objetivo para la función objetivo más relevante f_r sujeta a restricciones adicionales en las otras funciones objetivo. Los niveles ε_i se alteran después para generar otros puntos del conjunto Pareto. El método puede formularse de la siguiente manera:

1. Encontrar el óptimo de la r-ésima función objetivo; es decir, encontrar $\mathbf{x}^\star$ tal que:

$$f_r(\mathbf{x}^\star) = f_r^\star \tag{3.19}$$

   sujeta a restricciones adicionales de la forma

$$f_i(\mathbf{x}^\star) \leq \varepsilon_i \qquad \forall i \neq r \;\; (i, r \in \{1, \ldots, k\}) \tag{3.20}$$

   donde ε_i son los valores supuestos de las funciones objetivo que no deseamos exceder, en un contexto de minimización.

2. Repetir (1) para diferentes valores de ε_i. La información derivada de un conjunto bien seleccionado de ε_i puede ser útil para tomar la decisión final. La búsqueda se detiene cuando el tomador de decisiones encuentra una solución satisfactoria.

   Puede ser necesario repetir el proceso anterior para diferentes índices r. Para obtener valores adecuados de ε_i, se efectúan normalmente optimizaciones individuales para cada una de las funciones objetivo en turno, usando técnicas de programación matemática. El inconveniente obvio de esta técnica es que consume mucho tiempo y que la codificación de las funciones objetivo puede ser muy difícil o incluso imposible para ciertos problemas, particularmente si hay múltiples objetivos. Además, este método tiende a encontrar soluciones débilmente no dominadas, lo cual puede no ser muy adecuado en ciertas aplicaciones.

3.4. Discusión de los métodos clásicos

Los distintos métodos tradicionales que han sido presentados cuentan con al menos una de las siguientes dificultades [30, 32]:

1. El algoritmo de optimización se debe aplicar varias veces para encontrar un conjunto de soluciones Pareto óptimas. Como cada corrida es independiente de las demás, generalmente no se obtiene un efecto sinergético. Por tanto, delinear el frente Pareto óptimo resulta costoso en términos computacionales.

2. La mayoría de los algoritmos requieren conocimiento previo del problema a resolver y son muy sensibles a los parámetros del algoritmo: pesos de los objetivos, orden de evaluación, nivel de objetivos, valor de las restricciones, etc.

3. Algunos algoritmos son sensibles a la forma del frente Pareto.

4. La variación entre las diferentes soluciones encontradas depende de la eficiencia del optimizador de un sólo objetivo. Podría darse el caso de encontrar siempre la misma solución o soluciones muy parecidas en distintas corridas.

5. Relacionado al punto anterior, tenemos que en problemas que involucren al azar o incertezas, los métodos clásicos no son necesariamente confiables.

Investigaciones recientes han demostrado que las dificultades arriba mencionadas pueden ser superadas con la utilización de algoritmos evolutivos [30, 32]. En el siguiente capítulo se presentan estos algoritmos aplicados a optimizaciones con objetivos múltiples.

Capítulo 4

Algoritmos Evolutivos Multiobjetivo

La falta de métodos determinísticos eficientes y eficaces para la resolución de problemas con objetivos múltiples, motivó la búsqueda de métodos alternativos. El notable éxito obtenido por los algoritmos evolutivos en optimización monobjetivo y las características propias de los mismos despertaron el interés de los investigadores en utilizarlos en optimización multiobjetivo. En la actualidad, la optimización evolutiva multiobjetivo (*Evolutionary Multiobjective Optimization* -EMOO) es un área de investigación muy importante tanto para científicos como para ingenieros, no sólo por que la mayor parte de los problemas consideren por naturaleza objetivos múltiples, sino también porque aún quedan por resolver un sin número de interrogantes en esta disciplina. El interés creciente por este tipo de técnicas se refleja en la figura 4.1 que muestra el número de publicaciones por año, disponibles en [16], hasta mediados de 2003.

Si bien la noción de búsqueda genética para la exploración de soluciones óptimas en problemas con varios objetivos se remonta a fines de la década de los 60s, los primeros algoritmos evolutivos que consideraban de forma simultánea objetivos múltiples se desarrollaron recién a inicios de los noventa. Como una muestra de lo incipiente del área, el primer congreso internacional de EMOO se realizó en marzo del 2001 [100]. A pesar de esto, en poco tiempo, la computación evolutiva multiobjetivo, se ha

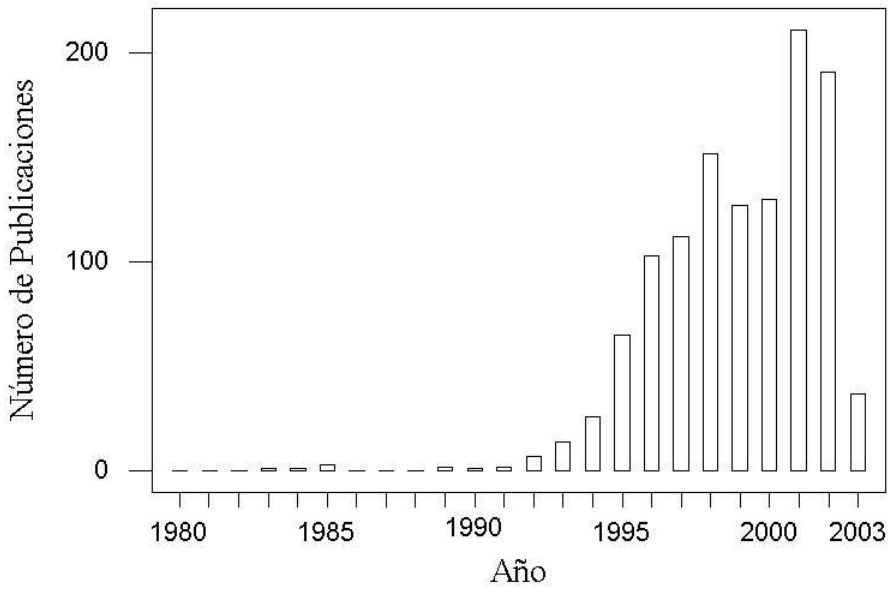

Figura 4.1. Publicaciones en optimización evolutiva con objetivos múltiples por año hasta mediados del 2002.

establecido como *el* método para aproximar el frente Pareto-óptimo en problemas de este tipo. Esto se debe fundamentalmente al paralelismo intrínseco de los algoritmos evolutivos que les permite explorar similaridades entre las soluciones de forma eficiente y a la capacidad de capturar varias soluciones Pareto-óptimas en una única corrida de optimización [99].

Los diferentes métodos para trabajar con objetivos múltiples utilizando algoritmos evolutivos se pueden clasificar, en forma sencilla, en técnicas de primera y segunda generación. Pertenecen a la primera generación las propuestas iniciales que no consideran conceptos de Pareto. Así mismo, ésta generación abarca a los primeros algoritmos evolutivos multiobjetivo basados en Pareto que no incluyen mecanismos para la preservación de las buenas soluciones encontradas durante el proceso evolutivo (elitismo). La segunda generación esta caracterizada básicamente por algoritmos basados en Pareto y que incorporan alguna forma de elitismo.

Puesto que los algoritmos genéticos requieren información escalar sobre el valor de adaptabilidad de los individuos, no es extraño que los primeros enfoques evolutivos para lidiar con objetivos múltiples se basen en la idea de combinar un algoritmo genético simple con métodos de escalarización de la función objetivo. Así, estos primeros enfoques se encargaban de optimizar la función agregada en vez de optimizar la verdadera función multiobjetivo [22, 44, 86].

En [76], Schaffer presentó el primer EA que no utiliza funciones de agregación para resolver problemas multiobjetivo [44], al que llamó *Vector Evaluated Genetic Algorithm* (VEGA). Este algoritmo utiliza un procedimiento evolutivo que básicamente divide una población genética en subpoblaciones, en las que se considera el valor de adaptación de los individuos de acuerdo a un objetivo distinto para cada subpoblación y realiza el cruzamiento mezclando individuos de distintas subpoblaciones. El resultado final del procedimiento de selección del VEGA corresponde a promediar los valores de cada uno de los objetivos [44]. A pesar que Schaffer tuvo algún éxito, especialmente para la resolución de problemas en aprendizaje de máquina [77], el método propuesto resultó ineficiente para explorar espacios de objetivos no convexos comportándose bien sólo en una dimensión [22].

En los años siguientes a la presentación de VEGA, se publicaron muy pocos trabajos en el área de optimización evolutiva para problemas con objetivos múltiples, los cuales tampoco lograron el éxito esperado. Estos trabajos proponían, por lo general, alguna combinación de métodos tradicionales con algoritmos evolutivos [44, 86]. Por ejemplo, en [47] se presenta un algoritmo que utiliza un método de selección basado en ordenamiento lexicográfico, donde pares de individuos se comparan de acuerdo a un objetivo prioritario, en caso de empate se considera el segundo objetivo de mayor prioridad, y así para cada objetivo.

Un renacimiento en el campo se produjo en la década de los noventa con la aparición de los primeros algoritmos que consideraban de forma simultánea la optimización de objetivos múltiples utilizando el concepto de dominancia Pareto. El *Multiobjective Genetic Algorithm* (*MOGA*) de Fonseca

y Flemming [42, 43], el *Niched Pareto Genetic Algorithm* (*NPGA*) de Horn y Nafpliotis [54, 55], y el *Nondominated Sorting Genetic Algorithm* (*NSGA*) de Srinivas y Deb [81, 82, 83] fueron los primeros MOEAs basados en Pareto. Estos algoritmos, pertenecientes a la primera generación de algoritmos evolutivos multiobjetivo, fueron aplicados a una amplia gama de problemas multiobjetivo, donde demostraron un mejor desempeño que el de los enfoques no basados en Pareto.

A pesar del éxito obtenido por los MOEAs de la primera generación basados en Pareto, una vez que estos encontraban una solución nodominada en una generación, éstas podían perderse cuando se aplicaban los operadores genéticos en las sucesivas generaciones. Para evitar la pérdida de buenas soluciones, el concepto de *elitismo*, utilizado en algoritmos monobjetivo, se extendió al campo de la optimización evolutiva multiobjetivo considerando la existencia de múltiples soluciones posibles. Así, los MOEAs de segunda generación están caracterizados por la utilización de conceptos de Pareto conjuntamente con alguna forma de elitismo para la búsqueda de soluciones. Algunos algoritmos correspondientes a la segunda generación son: el *Strenght Pareto Evolutionary Algorithm* (*SPEA*) de Zitzler y Thiele [102, 103], el *NSGA-II* de Deb, Agrawal, Pratab y Meyarivan [33], y el *Controlled Elitist NSGA-II* (*CNSGA-II*) de Deb y Goel [34], entre otros.

Durante la ejecución de un MOEA basado en Pareto un conjunto de soluciones Pareto óptimas con respecto a la población genética actual es encontrado en cada generación. A dicho conjunto se le denomina $P_{current}(t)$, donde t representa el número de generaciones transcurridas desde el inicio del procedimiento de evolución. Mientras que el frente Pareto correspondiente a $P_{current}(t)$ se denota como $PF_{current}(t)$. El conjunto de soluciones obtenidas al final de la ejecución de un MOEA basado en Pareto, esto es, el conjunto Pareto conocido, se denota con P_{known}. La notación utilizada para el frente Pareto asociado es PF_{known} respectivamente.

Puesto que los MOEAs se implementan sobre un modelo computacional finito (una computadora), los conjuntos generados son finitos. El conjunto Pareto "verdadero" para el modelo computacional se

denota con P_{true} y su correspondiente frente Pareto Optimo como PF_{true}. Los diferentes conjuntos generados deben ser suficientemente fieles al modelo matemático original, utilizado en el planteamiento del MOP. Cuando se resuelve un MOP utilizando MOEAs, la suposición implícita es que se cumple con al menos una de las siguientes relaciones: $P_{known} = P_{true}$, $P_{known} \subset P_{true}$ o $P_{known} \approx \mathcal{P}^{\star}$ [89]. Donde $\mathcal{P}^{\star}$ representa el frente Pareto óptimo teórico real (Definición 3.10).

En las secciones que siguen se presentan los MOEAs los algoritmos de primera y segunda generación utilizados en el presente trabajo para la realización de estudios experimentales.

4.1. MOEAs de primera generación

4.1.1. Multiobjective Genetic Algorithm

El MOGA (*Multiobjective Genetic Algorithm*) fue propuesto en [42, 43] como un algoritmo genético multiobjetivo basado en una modificación al algoritmo genético simple en la etapa de selección. El algoritmo se basa en un método de clasificación (*ranking*) de los individuos de la población genética de acuerdo al número de elementos que lo dominan. A partir de la información del procedimiento de clasificación, a cada individuo se le asigna un valor de adaptación. Luego, se utiliza un mecanismo de formación de nichos para evitar la convergencia prematura y mantener la diversidad.

El procedimiento de clasificación utilizado en MOGA asigna en cada generación a todos los elementos de la población genética un *ranking* igual al número de elementos que lo dominan más uno. Esto es, denotando como $p_i^{(t)}$ al número de individuos en la población que dominan a un individuo $P[i](t)$ y como $P[i](t)_{rank}$ a su *ranking* para todo $i \in \{1,\ldots,N\}$, el procedimiento de clasificación del

MOGA hace:

$$P[i](t)_{rank} = 1 + p_i^{(t)}$$

siendo:

$$p_i^{(t)} = ||\{\ j\ \mid\ j \in \{1,\ldots,N\} \wedge P[j] \succ P[i]\ \}||_c\ , \tag{4.1}$$

donde:

$||.||_c$ denota cardinalidad

Todos los elementos nodominados de la población $P(t)$ recibirán una clasificación de 1, mientras los dominados tendrán una clasificación > 1. En este esquema de clasificación, no todos los niveles están necesariamente representados en la población en una generación particular. Clasificados todos los elementos de la población, la asignación de *fitness* en MOGA se realiza de la siguiente manera:

1. Ordenar la población de acuerdo a la clasificación.

2. Asignar valor de adaptación a los individuos interpolando de la mejor clasificación a la peor de todas, de acuerdo a alguna función (usualmente lineal).

3. Promediar el valor de adaptación de los individuos con la misma clasificación, de tal forma que todos puedan ser muestreados a la misma tasa. Este procedimiento mantiene el *fitness global* de la población constante mientras que se mantiene una presión de selección apropiada.

Debido a errores aleatorios en el proceso de selección, cuando se tienen múltiples óptimos equivalentes, las poblaciones finitas convergen hacia una sola de ellas [42, 45]. Este fenómeno ha sido observado en algoritmos genéticos utilizados para la optimización de funciones multimodales monobjetivo. Para evitarlo, se desarrollaron e implementaron con éxito las técnicas de distribución del valor de adaptación y los métodos de formación de nichos (sección 2.7). En MOGA se utiliza una extensión de estas técnicas para la optimización multiobjetivo implementando *sharing* en el espacio objetivo,

utilizando la ecuación 2.8 entre pares de elementos nodominados y norma infinita o euclidiana para el cálculo de distancia, a fin de evolucionar una representación uniformemente distribuida de la superficie de compromiso global.

El MOGA presentado en [42] utiliza restricción de apareamiento a fin de evitar la recombinación entre cromosomas de nichos diferentes (sección 2.7.2) como en [35]. Si bien dicha técnica demostró ser adecuada para problemas con objetivo único, la pobre efectividad de la misma para problemas multiobjetivo ha hecho que en trabajos posteriores no se haya continuado con su utilización [68, 97].

4.1.2. Nondominated Sorting Genetic Algorithm

El *Nondominated Sorting Genetic Algorithm* (NSGA) [81, 82, 83], al igual que el MOGA, difiere del algoritmo genético simple sólo en la manera en que procede la selección, específicamente en la manera de asignar el valor de adaptabilidad. Los operadores de cruzamiento y mutación permanecen sin modificaciones. Así mismo, tanto el NSGA como el MOGA se basan en la utilización de un método de selección por clasificación para enfatizar los puntos actuales nodominados y un método de *niching* [62, 63] para preservar la diversidad en la población. Sin embargo, el NSGA utiliza un procedimiento de clasificación que, a diferencia del utilizado en MOGA, no considera el número de elementos que dominan a un individuo para clasificarlo, sino su nivel de dominancia. Esto es, el NSGA utiliza en forma directa la idea de realizar un procedimiento de búsqueda de óptimos para optimización multiobjetivo basado en una clasificación según la nodominancia, conforme fuera presentado en [50].

El procedimiento de asignación del valor de adaptación basado en clasificación por nodominancia (*nondominated sorting procedure*) utilizado en NSGA se presenta en el Algoritmo 3. En este procedimiento, primeramente se identifican los elementos nodominados de la población considerada, estos constituyen el primer frente nodominado de la población. Para ello, inicialmente, se marcan todos los elementos de la población como nodominados. En el Algoritmo 3, se denota con un subíndice *dom*

Algoritmo 3 Procedimiento de asignación de valor de adaptación en NSGA.

Procedimiento: fitness_assignement_nsga
Entrada : Población genética P
Salida : Población genética P con el valor de adaptación asignado a cada individuo

$\forall\, P[i] \in P;\, P[i]_{dom} = falso$ //marcar todos los elementos de la población como no dominados
$f = 0$ // hacer la cuenta de frentes igual a 0
$df = N$ //valor inicial del *dummy fitness* igual al tamaño de la población P
while $||P||_c > 0$ **do** //mientras queden elementos sin considerar
  $f = f + 1$ //incrementar la cuenta de frentes
  for $i = 1$ to $||P||_c$ **do** //para cada elemento $P[i]$ de P
    for $j = 1$ to $||P||_c$ **do** //para cada elemento $P[j]$ de P
      if $P[j] \succ P[i]$ **then** //si el elemento $P[j]$ domina a $P[i]$
        $P[i]_{dom} = true$ //marcar elemento $P[i]$ como dominado
        break //salir del ciclo
      end if
    end for
  end for
  $\mathcal{F}^f = \{P[i] \mid P[i]_{dom} = falso\}$ //los elementos de P no marcados conforman el frente f
  $\forall \mathcal{F}[i]^f \in \mathcal{F}^f;\, \mathcal{F}[i]^f_{fitness} = df$ //asignar a cada elemento del frente f un *fitness* grande
  fitness_sharing_procedure$(\mathcal{F}^f)$ //aplicar sharing a los elementos del frente $\mathcal{F}^f$
  $df = min\{\mathcal{F}[i]^f_{fitness} \mid \mathcal{F}[i]^f \in \mathcal{F}^f\}$ //guardar el peor valor de *fitness* del frente f en df
  $df = df - \varepsilon$ //reducir df en ε para usarlo como *dummy fitness* del siguiente frente
  $P = P \setminus \mathcal{F}^f$ //extraer de P el frente recientemente identificado reasignando los índices
  $\forall P[i] \in P;\, P[i]_{dom} = falso$ // marcar todos los elementos de la población como no dominados
end while
$P = \cup_{i=1}^{f} \mathcal{F}^i$ //recomponer la población P con los elementos de todos los frentes

a un campo del individuo que indica si es dominado o no. Además, se utiliza el sub-índice $objs$ para indicar el vector objetivo de un individuo. Inicialmente, a todos los individuos de la población se le asigna en su correspondiente campo *dom* un valor de falso, esto es, inicialmente todos son nodominados. Se inicializa a cero un contador de frentes f. Luego, el primer individuo de la población, $P[i]$ con $i = 1$, se compara con relación a la dominancia contra cada uno de los individuos de la población hasta encontrar un elemento que lo domine. Cuando esto ocurre, se marca el individuo $P[i]$ como dominado y se repite el procedimiento para el siguiente individuo en la población hasta que $i = N$.

Una vez que todos los individuos se comparan con respecto a la dominancia, los individuos no marcados son los nodominados con respecto a la población P. Estos individuos constituyen el primer frente de individuos nodominados, denotado por $\mathcal{F}^f$, con $f = 0$. Note que, en el peor de los casos,

cuando todos son nodominados, el procedimiento de clasificación presentado requiere N^2 comparaciones con respecto a la dominancia.

A todos los elementos del frente identificado se le asigna un valor de adaptación df, que en el caso del Algoritmo 3 inicialmente es igual al tamaño de la población genética N. Este valor de adaptación asignado a cada elemento del frente identificado, es un valor de adaptación ficticio (*dummy fitness*) que provee a todos los individuos del primer frente de un igual potencial reproductivo [81]. Con el fin de mantener la diversidad en la población, el valor de adaptación asignado a cada individuo del primer frente se comparte entre los elementos que componen dicho frente, dividiéndolo por la cuenta del nicho en el frente. Para ello se implementa el procedimiento de *fitness sharing* presentado en el Algoritmo 4 [81].

Determinado el primer frente de individuos nodominados de la población y finalizado el procedimiento de *fitness sharing*, se obtiene el peor valor de adaptación entre los individuos del primer frente y se guarda en df. Un valor ligeramente inferior a este será asignado como *dummy fitness* para los elementos del siguiente frente considerado, por lo que df se reduce en un $\varepsilon > 0$. Luego, los individuos del primer frente son eliminados de la población genética P de forma temporal. El procedimiento de clasificación se repite para los elementos de la población P sin los elementos del primer frente. Para ello, los elementos en P se vuelven a marcar todos como no dominados. El procedimiento continúa hasta que todos los frentes hayan sido identificados, es decir todos los elementos tengan asignado su correspondiente valor de adaptabilidad.

Finalizado el procedimiento de asignación del valor de adaptabilidad, la población es sometida al operador de selección utilizando el método de ruleta ponderada (sección 2.4.1) y a los demás operadores genéticos como es usual. Puesto que los elementos del primer frente tienen valores de adaptación mejores que los de cualquier otro frente, estos siempre obtienen más copias que el resto de la población. Así, este método dirige la búsqueda hacia las regiones de nodominancia, lo que final-

Algoritmo 4 Procedimiento de *fitness sharing* utilizado en NSGA.

fitness_sharing_procedure
Entrada: conjunto de individuos $\mathcal{F}^f$ del f-ésimo frente nodominado, radio de nicho σ_{share}.
Salida : conjunto de individuos $\mathcal{F}^f$ con *fitness* compartido asignado a cada elemento.
for $i = 1$ to $||\mathcal{F}^f||_c$ **do** `//desde` $i=1$ `hasta el número de elementos en el frente` f
  for $j = 1$ to $||\mathcal{F}^f||_c$ **do** `//desde` $j=1$ `hasta el número de elementos en el frente` f
    $d_{ij} = ||\mathcal{F}^f[i]_{objs} - \mathcal{F}^f[j]_{objs}||$ `//`d_{ij} `igual a dist. Euclid. entre objs. de` $\mathcal{F}^f[i]$ `y` $\mathcal{F}^f[j]$
    if $d_{ij} \leq \sigma_{share}$ **then** `// calcular valor de sharing (Ecuación 2.8 para` $P = \mathcal{F}^f$`)`
      $sh = 1 - (\frac{d_{ij}}{\sigma_{share}})^2$
    else
      $sh = 0$
    end if
    $\mathcal{F}^f[i]_{ncount} = \mathcal{F}^f[i]_{ncount} + sh$ `// calcular la cuenta de nicho para` $\mathcal{F}^f[i]$ `(Ecuación 2.9)`
  end for
  $\mathcal{F}^f[i]_{fitness} = \frac{\mathcal{F}^f[i]_{fitness}}{\mathcal{F}^f[i]_{ncount}}$ `//calcular el` *fitness compartido* `(Ecuación 2.10)`
end for

mente conduce al frente óptimo de Pareto.

4.1.3. Niched Pareto Genetic Algorithm

En [54, 55] se propone el *Niched Pareto Genetic Algorithm* (NPGA) el cual, como los anteriores, difiere del algoritmo genético simple únicamente en el procedimiento de selección. Sin embargo, este algoritmo no utiliza ningún procedimiento de clasificación por dominancia, sino que propone una variación del procedimiento de selección por torneo (sección 2.4.1) llamado torneo por dominancia Pareto y la utilización de un procedimiento de *sharing* para romper empates y determinar un ganador.

En los procedimientos de selección por torneo un conjunto de elementos son seleccionados en forma aleatoria de la población genética actual y el mejor de este conjunto es seleccionado. Estos procedimientos, diseñados para EAs aplicados en problemas monobjetivo, asumen que se desea una única solución al problema. Esto es, se desea que luego de cierto número de generaciones la población converja a una solución [54]. Para evitar esta convergencia a un único punto, en [54] se propone utilizar el operador de dominancia para la selección de individuos que compiten para ser seleccionados.

Si bien la relación de dominancia Pareto conduce de forma directa a un torneo binario, en el cual

dos individuos seleccionados de manera aleatoria se comparan de acuerdo a la domancia y aquel que domine al otro gana. Este tamaño de muestra es insuficiente para estimar que tan buena es una solución con respecto al conjunto. Por ello en [54] se propone un método distinto para la realización de la selección por torneo Pareto.

El Algoritmo 5 presenta el procedimiento de selección por torneo Pareto utilizado en el NPGA para seleccionar un conjunto de individuos, de tamaño igual al de la población genética, para la aplicación posterior de otros operadores genéticos conocido como conjunto para apareamiento (*matting pool*). En el procedimiento de selección del NPGA se seleccionan de la población genética de forma aleatoria dos individuos candidatos $P[i]$ y $P[j]$. También se selecciona de la población genética un conjunto de comparación P_{dom}. Luego, cada uno de los candidatos se compara contra cada uno de los elementos del conjunto comparación con relación a la dominancia. Si un candidato es dominado por el conjunto comparación y el otro no, se selecciona el nodominado. El tamaño del conjunto de comparación (t_{dom}) proporciona el control sobre la presión de selección, lo que se llama presión de dominancia (*domination pressure*).

Si ninguno de los candidatos o ambos son dominados por el conjunto de comparación, entonces se utiliza una forma simplificada de *sharing* para elegir un ganador. En este procedimiento de *sharing*, se calcula el número de individuos previamente seleccionados que se encuentran en las cercanías de cada solución candidata [55]. En caso que M se encuentre vacío, el cálculo de la cuenta de nicho se realiza sobre (P_{dom}). Debido al interés en mantener la diversidad, el candidato que tenga el menor número de individuos en su nicho es considerado como el mejor y por tanto seleccionado. Si el empate persiste, se elige uno de estos individuos de forma aleatoria.

Algoritmo 5 Procedimiento de selección por torneo Pareto, utilizado por el NPGA.

procedure: selection_npga
Entrada: Población P, radio de nicho σ_{share}, presión de dominancia t_{dom}.
Salida: Conjunto M de individuos seleccionados para la aplicación de otros operadores genéticos.
$M = \emptyset$, //La población M inicialmente se encuentra vacía
repeat
  Seleccionar aleatoriamente dos individuos $P[i], P[j] \in P$
  Seleccionar aleatoriamente un conjunto de comparación $P_{dom} \subseteq P$ con t_{dom} individuos.
  if $P[i]$ es nodominado con relación a P_{dom} y $P[j]$ es dominado **then**
    $M = M + \{P[i]\}$ //agregar el individuo $P[i]$ a M
  else if $P[j]$ es nodominado con relación a P_{dom} y $P[i]$ es dominado **then**
    $M = M + \{P[j]\}$ //agregar el individuo $P[j]$ al conjunto M
  else
    if $||M||_c = 0$ **then**
      //Calcular cuantos individuos en P_{dom} están a una distancia σ_{share} de $P[i]$ y $P[j]$
      $n_i = ||\{\ k \mid k \in \{1,\ldots,t_{dom}\} \wedge\ ||P[i]_{objs} - P_{dom}[k]_{objs}||_d\ \leq\ \sigma_{share}\}||_c$
      $n_j = ||\{\ k \mid k \in \{1,\ldots,t_{dom}\} \wedge\ ||P[j]_{objs} - P_{dom}[k]_{objs}||_d\ \leq\ \sigma_{share}\}||_c$
    else
      //Calcular cuantos individuos en M están a una distancia σ_{share} de $P[i]$ y $P[j]$
      $n_i = ||\{\ k \mid k \in \{1,\ldots,||M||_c\} \wedge\ ||P[i]_{objs} - M[k]_{objs}||_d\ \leq\ \sigma_{share}\}||_c$
      $n_j = ||\{\ k \mid k \in \{1,\ldots,||M||_c\} \wedge\ ||P[j]_{objs} - M[k]_{objs}||_d\ \leq\ \sigma_{share}\}||_c$
      if $n_i < n_j$ **then**
        $M = M + \{P[i]\}$ //si $P[i]$ tiene menos individuos en su radio de nicho agregarlo a M
      else if $n_j < n_i$ **then**
        $M = M + \{P[j]\}$ //si $P[j]$ tiene menos individuos en su radio de nicho agregarlo a M
      else
        $M = P[rand(j,i)]$ //si el empate persiste se selecciona en forma aleatoria
      end if
    end if
  end if
until $||M||_c = N$ //Hasta que el número de elementos en M sea igual al tamaño de P

La figura 4.2 representa el concepto de niching para romper empates en NPGA, para un problema de minimización de dos objetivos [38] cuando el *matting pool* se encuentra vacío. En dicha figura, se presenta un conjunto de individuos para la comparación y dos individuos candidatos (1 y 2). Ambos candidatos no nodominados con respecto al conjunto comparación. Para romper el empate, se debe considerar el número de individuos en el conjunto comparación que se encuentran en las cercanías de los individuos. Aquel individuo con menos "vecinos"será el ganador. En este caso, existen más individuos en las cercanías del candidato 1 y menos en las cercanías del candidato 2. Entonces, es el

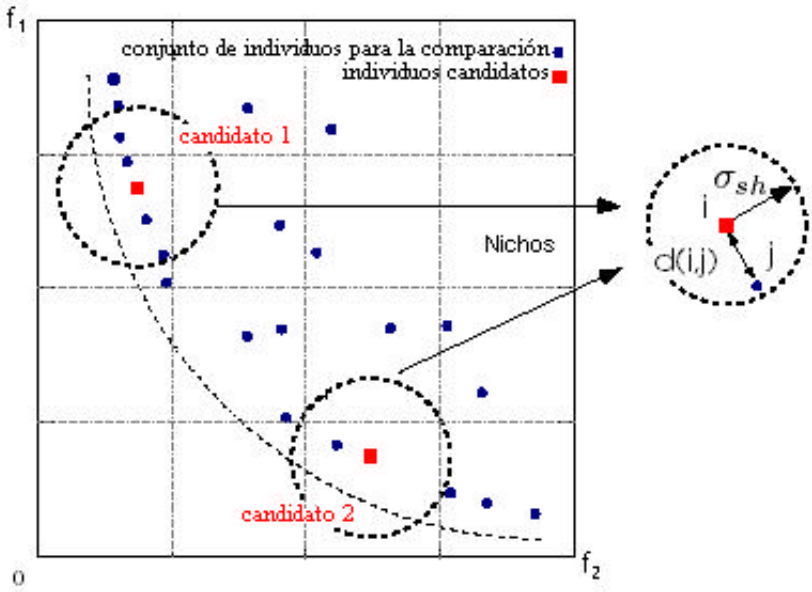

Figura 4.2. Concepto de Niching en NPGA.

candidato 2 el seleccionado para la posterior aplicación de otros operadores evolutivos.

4.2. MOEAs de segunda generación

4.2.1. Strength Pareto Evolutionary Algorithm

En [102, 103] se presenta un nuevo enfoque evolutivo para la optimización multiobjetivo, el *Strength Pareto Evolutionary Algorithm* (SPEA). Este algoritmo difiere de los anteriores en varios aspectos. Primeramente, utiliza dos poblaciones, incorporando el concepto de elitismo a través del almacenamiento de las soluciones nodominadas en una población externa, la cual participa del procedimiento de selección. Además, el cálculo del valor de adaptación se realiza utilizando un procedimiento basado en la asignación de un valor de fuerza (*strength*) a todos los elementos de la población externa. Este procedimiento induce la formación de nichos a partir del concepto de dominancia Pareto, llamado *niching* por *strength* [102]. Puesto que el conjunto de soluciones en la población externa puede ser grande y ésta interviene en el proceso evolutivo, también se utiliza un procedimiento de *clustering*

Algoritmo 6 Procedimiento principal del SPEA.

Paso 1 Generar una población inicial P y crear el conjunto nodominado externo $P' = \emptyset$.
Paso 2 Copiar los miembros nodominados de P a P'.
Paso 3 Eliminar las soluciones en P' cubiertas por cualquier otro miembro de P'.
Paso 4 Si el número de soluciones en el almacenamiento externo excede un máximo N', reducir P' por medio de *clustering* (Algoritmo 7).
Paso 5 Calcular el *fitness* de cada individuo en P, así como en P' (Algoritmo 8).
Paso 6 Seleccionar individuos de $P(t)+P'(t)$ (unión multiconjunto), hasta que el pool de apareamiento $P(t+1)$ se llene.
Paso 7 Aplicar los operadores de mutación y cruzamiento específicos del problema como es usual.
Paso 6 $t = t+1$
Paso 8 Si se alcanza el máximo número de generaciones parar, sino ir al Paso 2.

para reducir el número de soluciones en dicho conjunto.

El flujo principal del SPEA se muestra en el Algoritmo 6. Inicialmente se crea una población genética P de tamaño N y un conjunto externo de soluciones nodominadas P' vacío. Luego, las soluciones nodominadas de la población genética son copiadas al conjunto P'. De este conjunto se eliminan las soluciones cubiertas por cualquier otro miembro de dicho conjunto.

Definición 4.1. *Soluciones cubiertas en una población:* Dada una población P de tamaño N, para cualquier par de individuos en la población $P[i]$ y $P[j]$, $i, j \in \{1,\ldots,N\}$, se dice que $P[i]$ cubre a $P[j]$ si y sólo si: $i \neq j$ y $P[i] \succeq P[j]$. Es decir, si dados dos individuos en una población, con índices distintos i y j, el individuo en la posición i cubre al individuo en j si y sólo si considerando sus objetivos el individuo $P[i]$ es mejor o igual al individuo $P[j]$, esto es $P[i] \succeq P[j]$.

En ciertos problemas, el conjunto Pareto puede ser extremadamente grande, incluso infinito. Desde el punto de vista del tomador de decisiones, recibir todas las soluciones nodominadas encontradas es poco útil cuando el número de éstas excede los límites razonables. Más aún, el tamaño del conjunto externo nodominado influencia el comportamiento del SPEA. Por un lado, debido a que P' participa en la selección, demasiadas soluciones nodominadas podrían reducir la presión de selección y hacer la búsqueda más lenta. Por otro lado, el mecanismo de *niching por strength* se basa en una disposición uniforme de los puntos en P'. Si estos puntos no están distribuidos en forma uniforme, el proceso de

Algoritmo 7 Procedimiento de *clustering* para el SPEA.

Paso 1 Inicializar el *conjunto de clusters* C; cada punto nodominado externo $P'[i] \in P'$ constituye un cluster distinto, C es el conjunto unión :

$$C = \bigcup_{i}^{||P'||_c} \{P'[i]\}$$

Paso 2 Si $||C||_c \leq N'$, es decir si el número de clusters en C es menor o igual a N', ir al Paso 5, sino ir al Paso 3.

Paso 3 Calcular la distancia de todos los posibles pares de clusters. La distancia d_{12} entre dos clusters c_1 y $c_2 \in C$ está dada por la distancia promedio entre pares de individuos pertenecientes a los dos clusters.

Paso 4 Determinar dos clusters c_1 y c_2 con distancia mínima y amalgamarlos en un cluster mayor: $C = C \setminus \{c_1, c_2\} \cup \{c_1 \cup c_2\}$. Es decir se eliminan c_1 y c_2 del conjunto de clusters y se agrega a C un nuevo cluster formado de la unión c_1 con c_2. Ir al Paso 2.

Paso 5 Computar el conjunto nodominado reducido seleccionando de cada cluster el punto con distancia mínima a todos los otros puntos en el cluster considerado.

asignación de *fitness* posiblemente tenderá hacia ciertas regiones del espacio de búsqueda, conduciendo esto a una distribución desbalanceada de la población. Por tanto, si el número de elementos en el conjunto de nodominados supera un número máximo dado (N'), se procede a reducir el conjunto utilizando el procedimiento de *clustering* que se describe en el Algoritmo 7.

Luego, se calcula el valor de *fitness* utilizando el Algoritmo 8. Primero, a cada individuo del conjunto externo de nodominados P' se le asigna un valor real entre en $[0,1)$. Este valor indica la *utilidad* de un individuo y se le denomina su *strength*. El *strength* de un individuo $P'[i]$ está directamente relacionado con el número de elementos en la población genética para los cuales es mejor o igual. Esto es, dados dos individuos cualquiera en la población externa, el que domina de forma débil a más elementos en la población genética tiene el valor de *strength* mayor. El cálculo del valor del *strength* para un individuo en la posición *i* de la población externa P' es:

$$P'[i]_{strength} = \frac{||\{j | j \in \{1,\ldots N\} \wedge P'[i] \succeq P[j]\}||_c}{N+1} \tag{4.2}$$

El valor de adaptación del individuo $P'[i]$ será igual a la inversa de su valor de *strength*, esto es

$$P'[i]_{fitness} = \frac{1}{P'[i]_{strength}} \tag{4.3}$$

Luego de calcular el valor de adaptación de los individuos en la población externa, también basados en el cálculo del valor de *strength*, se calcula el valor de adaptación de los individuos en la población genética . El *strength* de un individuo $P[j] \in P$ se calcula a partir de los *strengths* de todas las soluciones externas no dominadas $P'[i] \in P'$ que lo dominen. Esto es:

$$P[j]_{strength} = 1 + \sum_{i, P'_i \succeq P_j} P'[i]_{strength} \tag{4.4}$$

Nuevamente, el valor de adaptación del individuo $P[j]$ será igual a la inversa de su valor de *strength*:

$$P[j]_{fitness} = \frac{1}{P[j]_{strength}} \tag{4.5}$$

Como resultado del procedimiento, los individuos en P' que cubren una cantidad menor de individuos en P reciben mayores valores de *fitness* que los otros miembros de la población. Los individuos que tienen muchos vecinos en el nicho son penalizados debido al alto valor del *strength* de los puntos nodominados asociados, esto es conocido como el *niching* por *strength*. La idea detrás de este mecanismo es preferir siempre los individuos que están más cerca del frente Pareto óptimo y al mismo tiempo, distribuirlos en toda la superficie factible. La principal diferencia de este método con respecto al *fitness sharing* es que el nicho no esta definido en términos de la distancia, sino de acuerdo a la dominancia de Pareto y no requiere el establecimiento de ningún parámetro predefinido. Esto es más adecuado ya que en muchos problemas del mundo real la distancia no tiene un significado práctico si se computa en el espacio objetivo, ya que cada objetivo puede estar expresado en magnitudes totalmente diferentes y no comparables a las de los demás (millones de dólares, segundos, metros, etc).

Finalizado el procedimiento de asignación del valor de adaptabilidad, se procede a la selección de los individuos para el cruce. Esta selección se realiza entre los elementos de ambas poblaciones. En [103] se utiliza un procedimiento de selección basado en torneo binario. Luego se aplican los operadores genéticos como es usual hasta que el número máximo de generaciones sea alcanzado.

Algoritmo 8 Algoritmo de asignación de *fitness* para el SPEA.

Paso 1 A cada solución $P'[i] \in P'$ se le asigna un valor real $P'[i]_{strength} \in [0,1)$, llamado su *strength*. $P'[i]_{strength}$ es proporcional al número de miembros de la población genética, $P[j] \in P$ para los cuales $P'[i] \succeq P[j]$. Sea N el número de individuos en P, entonces, el valor del *strength* para el individuo i de la población externa P' es:

$$P'[i]_{strength} = \frac{||\{j | j \in \{1,\ldots N\} \wedge P'[i] \succeq P[j]\}||_c}{N+1}$$

Esto es, el número de individuos en P que son cubiertos por $P'[i]_{strength}$ dividido el tamaño de la población genética más uno.

Paso 2 El valor de *fitness* de $P'[i]$ es:

$$P'[i]_{fitness} = \frac{1}{P'[i]_{strength}}$$

.

Paso 3 El *strength* de un individuo $P[j] \in P$ es calculado sumando los *strengths* de todas las soluciones nodominadas $P'[i] \in P'$ que lo cubren.

$$P[j]_{strength} = 1 + \sum_{i, P'_i \succeq P_j} P'[i]_{strength}$$

Paso 4 El valor de *fitness* de $P[j]$ es:

$$P[j]_{fitness} = \frac{1}{P[j]_{strength}}$$

.

4.2.2. Nondominated Sorting Genetic Algorithm II

En [33] se presentan los detalles del *Nondominated Sorting Genetic Algorithm II* (NSGA-II), un nuevo algoritmo basado en clasificación por nodominancia para asignar el valor de adaptabilidad a los elementos de la población genética. Son varias las características del NSGA-II que lo hacen diferente del NSGA original [81, 82]. En primer lugar, el NSGA-II incorpora un mecanismo de preservación de elites que asegura el mantenimiento de las buenas soluciones encontradas con anterioridad. En segundo lugar, el NSGA-II utiliza un procedimiento rápido de clasificación por nodominancia (*fast nondominated sorting procedure*) el cual incorpora un procedimiento especial de almacenamiento a fin de reducir la complejidad computacional del algoritmo presentado en [81]. Por último, a diferencia de su antecesor, el NSGA-II no requiere de ningún parámetro ajustable (ejemplo: σ_{share}) [33].

El Algoritmo 9 presenta el procedimiento rápido de clasificación por nodominancia sobre el cual basa su funcionamiento el NSGA-II. En este algoritmo primeramente se determinan, para cada solución $P[i]$ de una población P a clasificar:

1. El conjunto S_i de las soluciones dominadas por $P[i]$.

2. El número nd_i de soluciones que dominan a $P[i]$.

Para ello, se comparan entre si con respecto a la dominancia todos los miembros de la población a clasificar. Si un elemento $P[i]$ domina a un elemento $P[j]$, este último se agrega a un conjunto S_i. En caso contrario, si $P[j]$ es dominado por $P[i]$ entonces se incrementa el valor del contador nd_i de soluciones que dominan a $P[i]$. Note que un elemento $P[j]$ de la población puede pertenecer simultáneamente a nd_j conjuntos de soluciones dominadas.

Una vez determinados tanto el conjunto de soluciones dominadas como el número de soluciones que lo dominan, para cada elemento de la población, se forma el primer frente de soluciones nodominadas, denotado $\mathcal{F}^1$, con todos los elementos cuya cuenta de individuos que los dominan es igual a 0. El algoritmo prosigue recorriendo para cada elemento $P[i] \in \mathcal{F}^1$ su respectivo conjunto de soluciones dominadas S_i, reduciendo para cada elemento $P[j] \in S_i$ el valor de nd_j que le corresponde. Cuando el valor de nd_j se hace igual a 0, se agrega $P[j]$ a una lista $\mathcal{H}$ inicialmente vacía. Cuando se han recorrido todos los elementos del primer frente, en $\mathcal{H}$ quedan los elementos que sólo son dominados por los elementos del primer frente, es decir, los elementos del segundo frente $\mathcal{F}^2$. Luego se consideran los elementos de $\mathcal{F}^2$ repitiendo el procedimiento para cada elemento de dicho frente. El procedimiento finaliza cuando no quedan elementos cuya cuenta de elementos que lo dominan se haga cero, esto es cuando todos los frentes han sido identificados y $\mathcal{H} = \emptyset$.

El NSGA-II incluye también un procedimiento para estimar la densidad de soluciones alrededor de cada solución particular $\mathcal{F}^f[i]$ con respecto a los demás elementos del frente $\mathcal{F}^f$. El Algoritmo 10 ilus-

Algoritmo 9 Procedimiento Rápido de Ordenamiento por Nodominancia del NSGA-II.

Procedimiento :fast_nondominated_sort
Entrada : Una población P
Salida : Una lista de los frentes nodominados $\mathcal{F}$
for each $P[i] \in P$ **do**
  for each $P[j] \in P$ **do**
    if $P[i] \succ P[j]$ **then** //si $P[i]$ domina al elemento $P[j]$
      $S_i = S_i \cup P[j]$ //incluir $P[j]$ en S_i
    else if $P[j] \succ P[i]$ **then** //si $P[i]$ es dominado por $P[j]$
      $nd_i = nd_i + 1$ //incrementar nd_i
    end if
  end for
  if $nd_i = 0$ **then** //ninguna solución domina a $P[i]$
    $\mathcal{F}^1 = \mathcal{F}^1 \cup \{P[i]\}$ //$P[i]$ pertenece al primer frente
  end if
end for
$f = 1$ //el contador de frentes se hace igual a 1
while $\mathcal{F}^f \neq \emptyset$ **do**
  $\mathcal{H} = \emptyset$
  for each $P[i] \in \mathcal{F}^f$ **do** //para cada miembro $P[i]$ en $\mathcal{F}^f$
    for each $P[j] \in \mathcal{S}_i$ **do** //modificar cada miembro del conjunto S_i
      $nd_j = nd_j - 1$ //decrementar nd_j en uno
      if $nd_j = 0$ **then** //si nd_j es cero
        $\mathcal{H} = \mathcal{H} \cup \{P[j]\}$ //$P[j]$ es dominado sólo por elementos de $\mathcal{F}^f$
      end if
    end for
  end for
  $f = f + 1$ //se incrementa el contador de frentes
  $\mathcal{F}^f = \mathcal{H}$ //el frente actual está formado con los elementos de $\mathcal{H}$
end while

tra dicho procedimiento. En éste, para cada individuo $\mathcal{F}^f[i]$, se calcula un valor denotado $\mathcal{F}^f[i]_{distance}$ que sirve como un estimador del tamaño del cuboide más grande que encierra la solución sin incluir ningún otro punto de la población (a esto se le llama *crowding distance*). Para ello, se determina para cada individuo del frente considerado y para cada objetivo, cual es el siguiente menor y el siguiente mayor dentro de dicho frente. Esto se consigue ordenando de menor a mayor los elementos del frente para cada uno de los objetivos considerados. Luego, el valor de la distancia de *crowding* de un elemento $\mathcal{F}^f[i]$ se calcula sumando las distancias entre los individuos inmediatamente mayor y menor considerando cada objetivo. Note que los objetivos usualmente se encuentran expresados en unidades

Algoritmo 10 Procedimiento de asignación de distancia de *crowding* utilizado en NSGA-II.

Procedimiento: crowding_distance_assignment
Entrada: un conjunto de soluciones $\mathcal{F}^f$
Salida : el conjunto $\mathcal{F}^f$ con las distancia de *crowding* de sus elementos calculada
$l = ||\mathcal{F}^f||_c$
Para cada i, $\mathcal{F}^f[i]_{distance} = 0$ `//inicializar a cero la distancia de todos los elementos de` $\mathcal{F}^f$
for $j = 1$ to k **do** `//para cada objetivo considerado`
  Ordenar $\mathcal{F}^f$ de acuerdo al objetivo j
  $\mathcal{F}^f[1]_{distance} = \mathcal{F}^f[l]_{distance} = \infty$
  for $i = 2$ to $(l-1)$ **do**
    $\mathcal{F}^f[i]_{distance} = \mathcal{F}^f[i]_{distance} + (\mathcal{F}^f[i+1]_{objetivo[j]} - \mathcal{F}^f[i-1]_{objetivo[j]})$
    `// El sub-índice` $objetivo[j]$ `representa al valor del objetivo` j
  end for
end for

diferentes por lo que para obtener una estimación correcta es conveniente la normalización de los diferentes objetivos.

Además de definir un procedimiento de asignación de *crowding*, se define también un operador de comparación por *crowding* ($\geq_n$). El objetivo de este operador es guiar el proceso de selección en las diferentes etapas del algoritmo hacia un frente Pareto óptimo uniformemente distribuido.

Definición 4.2. Operador de *Crowding* **($\geq_n$):** Asumiendo que cada uno de los individuos en la población tiene dos atributos: la posición en la clasificación por nodominancia ($P[i]_{rank}$) y su distancia local de *crowding* ($P[i]_{distance}$), se define el orden parcial $\geq_n$ como:

$$P[i] \geq_n P[j] \text{ si } (P[i]_{rank} < P[j]_{rank}) \text{ o } ((P[i]_{rank} = P[j]_{rank}) \text{ y } (P[i]_{distance} > P[j]_{distance}))$$

donde $P[i]_{rank} = f$ si $P[i] \in \mathcal{F}^f$.

Esto es, se define un orden lexicográfico con dos objetivos, con la posición en la clasificación por nodominancia como el de mayor importancia. Entonces, entre dos soluciones con diferente posición en la clasificación por nodominancia se prefiere aquella con la clasificación más baja. De otra forma, si ambas soluciones están localizados en el mismo frente, se prefiere la que está ubicada en una región con un menor número de puntos.

Algoritmo 11 Algoritmo del NSGA-II.

$t = 0$
Generar una población $P(t)$ de tamaño N en forma aleatoria
Utilizar el Algoritmo 9 para obtener una lista $\mathcal{F}$ con los frentes de $P(t)$
Asignar a cada elemento de $P(t)$ un valor de inadaptabilidad igual a su nivel de nodominancia
Utilizar torneo binario para seleccionar elementos de $P(t)$ de acuerdo con su inadaptabilidad
Efectuar cruzamiento y mutación para producir una población hijo $Q(t)$ de tamaño N
while El criterio de parada no se cumpla **do**
  $f = 1$ `//hacer la cuenta de frentes igual a 1`
  $R(t) = P(t) \cup Q(t)$ `//combinar la población padre e hijo`
  $\mathcal{F}$=**fast_non_dominated_sort**($R(t)$) `//Usar Alg. 9 para obetner los frentes de` $R(t)$
  while $||P(t+1)||_c < N$ **do** `//Hasta llenar la población padre`
    crowding_distance_assignment($\mathcal{F}^f$) `//Calcular la distancia de` *crowding* `en` $\mathcal{F}^f$ `(Alg. 10)`
    $P(t+1) = P(t+1) \cup \mathcal{F}^f$ `//incluir el` f`-ésimo frente en la población padre`
    $f = f + 1$
  end while
  Ordenar de forma descendente de acuerdo al operador $\geq_n$
  Tomar los primeros N elementos de $P(t+1)$
  Seleccionar individuos de $P(t+1)$ utilizando torneo binario de acuerdo al operador de *crowding*
  Aplicar cruzamiento y mutación sobre los individuos seleccionados para obtener $Q(t+1)$
  $t = t + 1$
end while

Presentados todas las partes que conforman el NSGA-II, en el Algoritmo 11 se presenta el bloque principal del mismo. Inicialmente se genera una población padre $P(0)$ de tamaño N. Esta población se clasifica en base a la nodominancia utilizando el Algoritmo 9. Terminado el proceso de clasificación, se asigna a cada solución un valor de inadaptabilidad igual a su nivel de nodominancia. Luego, se utilizan los operadores de selección por torneo binario, cruzamiento y mutación para generar, a partir de $P(0)$, una nueva población hijo $Q(0)$ también de tamaño N. Obtenidas las poblaciones padre e hijo iniciales, se realizan las sucesivas generaciones mientras la condición de parada no se cumpla.

En cada generación, se forma una población combinada $R(t) = P(t) \cup Q(t)$. Esto permite que las soluciones padres sean comparadas con la población hijo, asegurando el elitismo. Luego, la población $R(t)$, de tamaño $2N$, se clasifica de acuerdo a la nodominancia. Para ello se utiliza de nuevo el algoritmo de ordenamiento rápido por nodominancia, con el cual se identifican los diferentes frentes nodominados existentes en $R(t)$. Una vez que se obtienen todos los frentes, la nueva población padre

$P(t+1)$ se forma agregando soluciones del primer frente $\mathcal{F}^1$, continuando con los demás frentes hasta que el tamaño exceda o sea igual a N. Luego, la población $P(t+1)$ se ordena de acuerdo al operador de *crowding* y se forma la población $P(t+1)$ con los primeros N elementos. Los individuos de cada frente se utilizan para calcular la distancia entre las soluciones vecinas (distancia de *crowding*) utilizando el procedimiento **crowding_distance_assignment** (Algoritmo 10). Las soluciones del último frente aceptado es ordenado de acuerdo a un criterio de comparación de *crowding* y se toman los elementos de este frente hasta que completar el total de N soluciones en P. Finalmente, los elementos en $P(t+1)$ son utilizados para crear una nueva población $Q(t+1)$ utilizando selección, cruzamiento y mutación. A diferencia del procedimiento de selección por torneo binario usual, se utiliza un procedimiento que considera el operador de *crowding* para la competencia entre individuos. El procedimiento se repite hasta que el número máximo de generaciones u otro criterio de parada sea alcanzado.

4.2.3. Controlled Elitist NSGA-II

El algoritmo NSGA-II con elitismo controlado (*Controlled Elitist NSGA-II - CNSGA-II*) [34] difiere del NSGA-II únicamente en que el número de individuos pertenecientes al mejor frente actual de nodominados es seleccionado de forma adaptativa. Durante el proceso evolutivo, en la población combinada $R(t)$ del NSGA-II, podrían existir demasiadas soluciones en el primer frente de nodominados, las cuales pueden estar lejos del frente verdadero. Debido a la preponderancia de elementos en el primer frente de nodominados, la información genética de los elementos pertenecientes a otros frentes puede perderse. En varios problemas, la pérdida de *diversidad lateral* puede producir una reducción en la velocidad de convergencia del EA, en especial en problemas multimodales donde las poblaciones pueden ser atraídas por frentes Pareto-óptimos locales [34].

Para evitar la pérdida de información genética correspondiente a los distintos frentes, en el algoritmo CNSGA-II se intenta mantener una distribución predefinida de elementos de cada frente. En

consecuencia, se propone que el número máximo de elementos incorporados a cada nueva generación, pertenecientes al f-ésimo frente, η_f, se calcule de acuerdo a la distribución geométrica :

$$\eta_f = t_{red} \cdot \eta_{(f-1)} \tag{4.6}$$

donde $0 < t_{red} < 1$ es llamada la *tasa de reducción*. Si bien el procedimiento requiere que el parámetro t_{red} sea definido por el usuario, el procedimiento para conformar la nueva población a partir de elementos de distintos frentes es adaptativo.

Primeramente, la población $R(t) = P(t) \cup Q(t)$ se clasifica según la nodominancia. Siendo κ el número de frentes existentes en la población $R(t)$ (de tamaño $2N$), entonces, de acuerdo a la distribución geométrica, el número máximo de individuos permitidos provenientes del f-ésimo frente ($f = 1, \ldots, \kappa$) en la nueva población de tamaño N es :

$$\eta_f = N \frac{1 - t_{red}}{1 - (t_{red})^{\kappa}} \cdot (t_{red})^{f-1} \tag{4.7}$$

Puesto que $0 < t_{red} < 1$, el número máximo de individuos permitidos en el primer frente es más alto. Luego, de cada frente se permite un número exponencialmente decreciente de soluciones. Esta distribución exponencial es utilizada en [34] como una suposición que debe ser contrastada con otras distribuciones posibles. Sin embargo, el principio básico es siempre forzar la permanencia de soluciones provenientes de todos los frentes nodominados para que coexistan en la población.

La ecuación 4.7 denota el número máximo de individuos η_f de cada frente f de la población $R(t)$ que pasarán a formar parte de la población $P(t+1)$. Es probable que no existan exactamente η_f individuos en el frente f. Suponga que hay $\eta_f^t = ||\mathcal{F}^1(t)||_c$ individuos en el primer frente. Si $\eta_1^t > \eta_1$, esto es, si existen más soluciones que las permitidas, sólo se toman las primeras η_1 soluciones ordenadas de acuerdo al operador de *crowding*. De esta manera se seleccionan exactamente η_1 soluciones que están en las regiones menos pobladas. Por otro lado, si $\eta_1^t < \eta_1$ (esto es, si existen menos soluciones en la población que las permitidas), se eligen todas las soluciones en el frente y se cuenta el número

de espacios que quedaron vacíos. El máximo número de individuos en el segundo frente se incrementa a $\eta_2 = \eta_2 + (\eta_1 - \eta_1^t)$. Luego, el número actual de soluciones η_2^t presente en el segundo frente se encuentra y se compara con η_2 como se hizo con el frente anterior. Este procedimiento continúa hasta que todos los *N* individuos sean seleccionados. Fuera de este procedimiento de elitismo controlado, el resto del procedimiento es igual al del NSGA-II.

En ciertas situaciones es posible que a pesar de procesar las $2N$ soluciones como se indicó anteriormente, aún queden espacios vacíos en la nueva población. Esto pasa particularmente con valores grandes de t_{red}. En tal caso, se realiza otra pasada con los elementos restantes del primer frente, continuando con los otros frentes, hasta llenar la población completamente.

Capítulo 5

Computación Paralela + MOEAs = Algoritmos Evolutivos Paralelos Multiobjetivo

Probablemente la mayor fuerza que mueve la industria de las computadoras sea la creciente demanda por incrementar el poder de cómputo, ya que, a medida que la velocidad de las computadoras aumenta, la necesidad de poder computacional crece por lo menos al mismo ritmo [84]. A medida que son satisfechas las necesidades de cómputo para la aplicación actual, nuevas tecnologías son necesarias para cumplir con los requerimientos de nuevas aplicaciones que exigen cada vez más recursos [46, 84].

Para enfrentar problemas cada vez más complejosa, la potencia de cómputo que se pude obtener a partir de un único procesador es insuficiente. La velocidad de la luz y cuestiones sobre la física de los componentes imponen límites a la velocidad que se puede conseguir utilizando la típica *arquitectura de von Neumann*. El procesamiento paralelo ha surgido como la alternativa para satisfacer las características de velocidad de cómputo que las aplicaciones requieren [84].

Una computadora paralela es un conjunto de procesadores capaces de trabajar cooperativamente para resolver un problema computacional [46]. Esta definición es suficientemente amplia de forma a incluir supercomputadoras paralelas con cientos o miles de procesadores, conjuntos de estaciones de trabajo, estaciones de trabajo con varios procesadores, sistemas empotrados, etc. Las computadoras paralelas ofrecen la posibilidad de concentrar recursos computacionales en la solución de problemas computacionales importantes [46].

Los primeros algoritmos paralelos fueron escritos a comienzos de los sesentas, aunque aún no existían entornos paralelos. Los investigadores resolvían problemas asumiendo la existencia de un entorno de ejecución paralela, sin preocuparse de la aplicabilidad de sus investigaciones. Recién en los 70s aparecen las primeras computadoras paralelas [46, 84].

Muchos algoritmos para la resolución de problemas son paralelizables por naturaleza. Aunque a veces es claro que la solución a muchos problemas puede obtenerse fácilmente ejecutando en forma simultánea algunas etapas del proceso de resolución, una conversión sencilla de un algoritmo secuencial a un algoritmo paralelo puede no proveer el máximo paralelismo obtenible para un problema dado [46, 66]. El hecho de que no siempre es posible convertir el mejor algoritmo secuencial en el mejor algoritmo paralelo, esto es, no hay una correspondencia trivial entre computación paralela y secuencial hace necesario el estudio y desarrollo de técnicas de ingeniería para la construcción y el diseño de algoritmos paralelos [46].

La integración de técnicas de computación paralela con otras tecnologías permite el desarrollo de nuevas aplicaciones comerciales y científicas. En particular, la integración de la computación paralela y la computación evolutiva ha dado nacimiento a los algoritmos evolutivos paralelos multiobjetivo (*parallel Multiobjective Evolutionary Algorithms - pMOEAs*).

Los algoritmos evolutivos multiobjetivo han sido utilizados para la resolución de un gran número de MOPs como la optimización de potencia reactiva [5, 6], el calendarizado del encendido de bom-

bas en estaciones de bombeo de agua [92], diseño de redes de computadoras [37], entre otros. Con el creciente interés en la utilización de MOEAs en problemas de optimización del mundo real, se hace necesario mejorar el desempeño de estos algoritmos tanto en la calidad de las aproximaciones obtenidas como en la velocidad del proceso de búsqueda. La utilización de conceptos de paralelismo en MOEA es una alternativa para conseguir estas mejoras, puesto que la utilización de un número mayor de procesadores (y memoria) trabajando en el problema, en principio, permite la exploración de un espacio de soluciones mayor en un periodo dado de tiempo [88, 89].

La utilización de pMOEAs, para la resolución de problemas de optimización multiobjetivo, posee varias ventajas con respecto a otros métodos uniendo las características propias de los MOEAs con las ventajas del cómputo paralelo. Básicamente, los pMOEAs buscan encontrar soluciones tan buenas o mejores que sus contrapartes secuenciales en menos tiempo y/o explorar un espacio mayor de posibles soluciones [89]. Es decir, siendo fácilmente paralelizables, los pMOEAs ofrecen al menos una de las siguientes ventajas adicionales, con relación a otras técnicas de resolución de MOPs [89]:

- capacidad de explorar espacios de búsqueda complejos y de alta dimensionalidad;
- capacidad de explorar espacios de búsqueda mayores que sus contrapartes secuenciales;
- posible reducción del tiempo total de ejecución;
- posible mejora en la calidad de las soluciones obtenidas.

En la siguiente sección se presenta un breve repaso a algunos conceptos sobre computación paralela que serán utilizados posteriormente en el desarrollo del presente trabajo.

5.1. Conceptos de Computación Paralela

5.1.1. Clasificación del procesamiento paralelo

Existen varias maneras de clasificar las arquitecturas de computadoras. Si bien no existe una clasificación que puede considerarse completa, la clasificación de Flynn es la más ampliamente difundida [84]. En [40], Flynn clasificó las arquitecturas de computadoras basado en los diferentes flujos usados en el proceso de computación. Un flujo es una secuencia de objetos tales como datos o de acciones como instrucciones [40]. La secuencia de instrucciones leída de la memoria constituye un flujo de instrucciones. Las operaciones ejecutadas sobre los datos en el procesador constituyen un flujo de datos. La clasificación de Flynn divide a las computadoras en cuatro grupos principales:

- flujo único de instrucciones y flujo único de datos (*Single Instruction stream Single Data stream - SISD*),
- flujo múltiple de instrucciones y flujo único de datos (*Multiple Instruction stream Single Data stream - MISD*),
- flujo único de instrucciones y flujo múltiple de datos (*Single Instruction stream Multiple Data stream - SIMD*)
- flujo múltiple de instrucciones y flujo múltiple de datos (*Multiple Instruction stream Multiple Data stream - MIMD*)

La máquina tradicional de von Neumann cae en la primera categoría. MISD es sólo de interés teórico ya que ninguna máquina pertenece al conjunto [84], mientras las máquinas paralelas se clasifican como SIMD (paralelismo en el flujo de datos) y MIMD (paralelismo en el flujo de instrucciones y datos).

MIMD es la forma más familiar de procesamiento paralelo, por lo que la literatura en general se refiere a éstas últimas cuando se habla de computadoras paralelas. Sin embargo, existen ciertas com-

putadoras paralelas que no se ajustan a está clasificación y pertenecen al conjunto de lo que algunos han llamado arquitecturas alternativas [8].

Los sistemas con un número reducido de unidades de procesamiento central (*Control Process Units - CPU*) grandes e independientes que tienen conexiones de baja velocidad entre sí se dice que están *débilmente acoplados*. Lo opuesto son los sistemas *fuertemente acoplados*, en los que los componentes generalmente son más pequeños, están más juntos e interactúan entre sí a través de redes de comunicación con gran ancho de banda.

El término *granularidad* se refiere al tamaño de la unidad que se paraleliza, o tamaño de grano. Así tenemos el *paralelismo de grano grueso*, cuando la ejecución de programas grandes en paralelo con poca o ninguna comunicación entre los programas y su opuesto *el paralelismo de grano fino*. En general, los problemas que tienen paralelismo de grano grueso se resuelven mejor en sistemas débilmente acoplados mientras que los problemas con paralelismo de grano fino se resuelven mejor en sistemas fuertemente acoplados [39, 46, 84]. No obstante, es tanta la variedad de algoritmos, software y hardware, que en el mejor de los casos ésta es sólo una guía general [84].

Existen dos modelos muy difundidos de computadoras paralelas: los multiprocesadores y las multicomputadoras. Un multiprocesador es un sistema compuesto por múltiples procesadores donde cada procesador individual tiene su propia unidad de control y ejecuta su propio programa, pero los procesadores comparten el mismo sistema de memoria [8]. En las multicomputadoras, cada procesador tiene su propia memoria local privada, accesible únicamente a ella y a ninguna otra CPU, por lo que se les suele denominar sistemas de memoria distribuida.

Los procesos que trabajan en diferentes partes del mismo problema se deben comunicar entre sí de alguna forma para intercambiar información. En los multiprocesadores, ya que todas las CPUs comparten una misma memoria física, cualquier proceso puede leer o escribir una palabra de memoria con sólo ejecutar una instrucción de lectura o escritura. Todos los procesos que están ejecutándose en

un multiprocesador pueden compartir un sólo espacio de direcciones virtuales mapeado en la memoria común. Dos procesos pueden comunicarse con sólo hacer que uno de ellos escriba datos en la memoria y que el otro los lea después.

En las multicomputadoras, en cambio, cada CPU tiene su propia memoria local privada a la que puede acceder de forma directa, pero a la que ninguna otra CPU puede acceder directamente. Es decir, no tienen un método directo de comunicación entre procesadores. Para compartir información es preciso un mecanismo de paso de mensajes (o algún otro) que permita el intercambio de datos entre distintos procesadores, lo que adhiere ciertas complicaciones al diseño y la escritura de programas para este tipo de máquinas [46, 84].

Mientras que la programación en multiprocesadores es algo mucho más simple que en una multicomputadora, éstos son difíciles de construir y no pueden escalarse a tamaños muy grandes, además, la contención por la memoria en un multiprocesador puede afectar mucho el desempeño. Las multicomputadoras, sin embargo, son relativamente simples de construir [84].

Debido a la dificultad en la construcción de multiprocesadores con un gran número de procesadores, los esfuerzos por obtener mayor poder de cómputo en la actualidad se dirigen hacia la construcción y uso de multicomputadoras [66, 46, 84]. En el caso de las multicomputadoras, es difícil dar una taxonomía nítida, no obstante, destacan dos estilos generales: los procesadores masivamente paralelos (*Massively Parallel Processors - MPP*), y las asociaciones de estaciones de trabajo (*Cluster Of Workstations - COW*) [84].

Los MPP son poderosas supercomputadoras con un costo de varios millones de dólares, siendo actualmente las computadoras más poderosas del mundo. En lo esencial, un MPP es una colección de elementos de computación más o menos estándar unidos por un sistema de interconexión muy rápido. Estos se usan en la ciencia, ingeniería y la industria para cálculos de gran envergadura, para manejar números enormes de transacciones por segundo, o para el almacenamiento y gestión de grandes bases

de datos. Un ejemplo de MPP es la Intel/Sandia Option Red [84], construida especialmente para el Departamento de Defensa y de Energía de EEUU, posee 4608 nodos, donde cada uno de estos es un Pentium Pro de 200 MHz y tiene un costo de alrededor de 80 millones de dólares.

A medida que el precio de los componentes de los sistemas de cómputo disminuye y las redes de área local se hacen más rápidas y confiables, aumenta el interés en unir múltiples computadoras para construir sistemas paralelos grandes y poderosos. Estos sistemas, consistentes en unos cuantos cientos de computadoras personales o estaciones de trabajo conectadas por una tarjeta de red comercial, han sido denominados *Cluster Of Workstations*. Existen dos tipos principales de COW: los centralizados y los descentralizados. Un COW centralizado es un conjunto de estaciones de trabajo o computadoras personales montadas en un compartimiento enorme en un solo recinto, mientras los COW descentralizados consisten en estaciones de trabajo dispersas dentro de un edificio o campus universitario. Por otro lado, los distintos tipos de COW pueden ser a su vez homogéneos o heterogéneos. En el primer caso todos los elementos de proceso comparten las mismas características tanto en software como en hardware, mientras que en el segundo caso existen diferencias entre los equipos interconectados en la red.

5.1.2. Cómputo distribuido

El cómputo distribuido es un proceso mediante el cual un conjunto de computadoras conectadas a través de una red es utilizado de forma colectiva para la resolución de un único problema, usualmente de gran tamaño. En un MPP, cada procesador es exactamente igual en cuanto a sus capacidades, recursos, software y velocidad de comunicación. Esto no ocurre en una red de computadoras, por lo que cuando un programador desea explotar una red de computadoras para la resolución de un problema, este debe enfrentarse con heterogeneidad en varios niveles (Ej. arquitectura, formato de datos, velocidad computacional, carga de máquina, carga de red) [39, 84] .

A pesar de las numerosas dificultades causadas por la heterogeneidad, el cómputo distribuido ofrece muchas ventajas [39, 84]:

- Debido al uso del hardware existente, el precio del cómputo puede ser bastante bajo.
- El desempeño puede ser optimizado si se asigna a cada tarea individual la arquitectura más adecuada.
- Ciertos problemas son inherentemente distribuidos.
- La posibilidad de crecer en forma incremental y de esta forma tomar ventaja de lo último en tecnología computacional y de red.
- Permite compartir recursos costosos.
- Las computadoras individuales son relativamente estables, y una gran experiencia en el uso ya está disponible.
- Se puede proveer con un mínimo de esfuerzo tolerancia a fallos a nivel de usuario o de programa.
- Una proporción precio-desempeño muy superior que la ofrecida por los MPP.

5.1.3. Herramientas de software para la comunicación en multicomputadoras

Para programar una multicomputadora se requiere software especial, por lo regular bibliotecas, que se encarguen de la comunicación entre procesos y la sincronización. Casi todos los sistemas de transferencia de mensajes cuentan con dos primitivas (por lo regular llamadas de biblioteca): enviar (*send*) y recibir (*receive*), pero puede haber varios tipos de semánticas distintas. Las tres principales variantes son :

1. Transferencia de mensajes sincrónica.

2. Transferencia de mensajes con *buffers*.

3. Transferencia de mensajes no bloqueadora (asíncrona).

En la transferencia de mensajes sincrónica, si el transmisor ejecuta un *send* y el receptor todavía no ha ejecutado un *receive*, el transmisor se bloquea (suspende) hasta que el receptor ejecuta un *receive*, y sólo en ese momento se copia el mensaje. Cuando el transmisor recupera el control, después de la comunicación, sabe que el mensaje se envió y se recibió correctamente. Este método tiene la semántica más sencilla y no requiere *buffers*. Una desventaja importante es que el transmisor permanece bloqueado hasta que el receptor ha obtenido el mensaje y ha confirmado tal recepción.

En la transferencia de mensajes con *buffers*, cuando se envía un mensaje antes de que el receptor esté listo, el mensaje se coloca en un *buffer* en algún lado, por ejemplo en un buzón, hasta que el receptor lo saca. Así, en la transferencia de mensajes con *buffers* un transmisor puede continuar después de un *send*, aunque el receptor esté ocupado con otra cosa. Puesto que el mensaje realmente se envió, el transmisor está en libertad de reutilizar su *buffer* de mensajes de inmediato. Este esquema reduce el tiempo que el transmisor tiene que esperar. Básicamente, tan pronto como el sistema ha enviado el mensaje, el transmisor puede continuar. Sin embargo, el transmisor no tiene ninguna garantía de que el mensaje se recibió correctamente. Incluso cuando la comunicación es confiable, el receptor podría haberse caído antes de recibir el mensaje.

En la transferencia de mensajes no bloqueadora o asíncrona, se permite al transmisor continuar de inmediato después de efectuar una comunicación. Lo único que la biblioteca hace es pedir al sistema operativo que lleve a cabo la comunicación después, cuando tenga tiempo. El resultado es que el transmisor prácticamente no se bloquea.

Dos sistemas de transferencia de mensajes que se usan en muchas multicomputadoras son: PVM (*Parallel Virtual Machine*) y MPI (*Message Passing Interface*) [39, 66]. Éstos no son los únicos, pero son los más ampliamente utilizados. PVM es el más antiguo y fue el estándar de facto durante años, por

lo que fue escogido para este trabajo. En lo que sigue, se señalan las características más importantes de esta librería.

PVM es un sistema de transferencia de mensajes de dominio público diseñado inicialmente para ejecutarse en COW basadas en Unix. Posteriormente se le ha trasladado a muchas otras plataformas, incluidos casi todos los MPP comerciales. PVM es un conjunto integrado de herramientas de software y librerías que emulan una máquina paralela flexible de propósito general sobre una colección de computadoras interconectadas. El objetivo del sistema PVM es permitir a una colección de computadoras ser utilizadas en forma cooperativa para computación paralela.

Los principios básicos sobre los cuales se basa PVM son:

- *Pool de host configurable por el usuario:* una tarea computacional se ejecuta sobre un conjunto de máquinas que pueden ser seleccionadas por un usuario. El conjunto de equipos puede ser alterado añadiendo y eliminando máquinas durante la operación.

- *Acceso transparente al Hardware:* los programas de aplicación pueden ya sea ver el entorno hardware como una colección de elementos de procesamiento sin atributos o puede elegir explotar las capacidades de máquinas específicas del conjunto poniendo explícitamente ciertas tareas en las computadoras más apropiadas.

- *Computación basada en procesos:* la unidad de paralelismo en PVM es una tarea, un hilo independiente de control que alterna entre comunicación y ejecución.

- *Modelo de paso de mensajes explícito:* colecciones de tareas computacionales, cada una realizando una parte de la carga de trabajo de la aplicación, cooperando gracias al envío de mensajes entre equipos en forma explícita.

- *Soporte para heterogeneidad:* el sistema PVM soporta heterogeneidad en términos de máquinas, redes y aplicaciones.

- *Soporte para multiprocesadores:* PVM usa las facilidades nativas de paso de mensaje sobre multiprocesadores para tomar ventaja del hardware subyacente.

Habiendo presentado un breve repaso de los conceptos sobre paralelismo más importantes, se está ahora en condiciones de presentar la manera en que la integración entre cómputo paralelo y computación evolutiva multiobjetivo se ha dado.

5.2. Algoritmos Evolutivos Paralelos Multiobjetivo

Si bien los conceptos de paralelismo en optimización evolutiva no son una idea reciente, estos apenas han sido integrados a la optimización multiobjetivo. De hecho, de las más de 1320 citas existentes en [16] menos del 4 por ciento de los trabajos corresponden a MOEAs paralelos (*parallel MOEAs - pMOEAs*) [89].

Las primeras publicaciones relativas a la paralelización de EAs se remontan a mediados de la década del 80 y en la actualidad existen varios modelos propuestos para el desarrollo de EAs paralelos. Los tres modelos más importantes son:

- el modelo maestro-esclavo (*master-slave model*),
- el modelo de difusión (*diffusion model*) y
- el modelo de islas (*island model*).

En [11] se presenta una descripción detallada de distintos modelos utilizados en optimización monobjetivo, mientras que recientemente [89] lo presenta en un contexto multiobjetivo.

5.2.1. Modelo maestro-esclavo

Para la mayor parte de los problemas multiobjetivo, el principal costo computacional viene dado por el cálculo de funciones objetivo complejas, las cuales pueden incluir, por ejemplo, la realización de un proceso de simulación como en [37]. Entonces, paralelizar el cómputo de esta función es una alternativa para obtener una mejora en el desempeño del algoritmo. Este es básicamente el enfoque maestro-esclavo, ilustrado en la Figura 5.1. Los esclavos realizan el cálculo de la función objetivo y el maestro se encarga de distribuir el cómputo entre los esclavos así como de la ejecución de los operadores evolutivos.

Al acelerar el cómputo de la función objetivo, el modelo maestro-esclavo posibilita la realización de más generaciones que un algoritmo secuencial en el mismo tiempo. Así, el modelo maestro esclavo permite la exploración de un espacio de búsqueda mayor en un tiempo dado, por lo que es de esperar que se obtengan mejores aproximaciones que utilizando un algoritmo secuencial. Sin embargo, aunque con este modelo es posible obtener una mejora en la velocidad de ejecución, el comportamiento global del algoritmo evolutivo multiobjetivo paralelo maestro-esclavo es el mismo que el de los algoritmos secuenciales.

La distribución de la evaluación de la función objetivo sobre un conjunto de procesadores esclavos generalmente se implementa de tres formas diferentes [89]:

1. Distribuyendo los miembros de la población, entre los esclavos donde cada esclavo realiza la

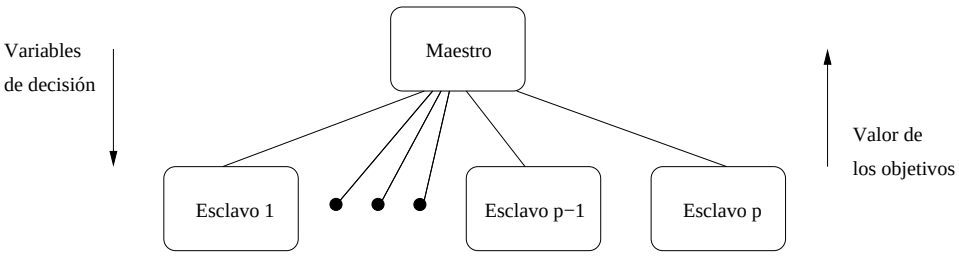

Figura 5.1. Modelo maestro esclavo.

evaluación de las *k* funciones objetivo.

2. Distribuyendo miembros de la población entre un conjunto de *k* esclavos donde cada uno realiza la evaluación de uno de los objetivos.

3. Distribuyendo el cálculo de la función objetivo en sí misma para toda la población entre varios procesadores.

En el primer método, cada procesador esclavo calcula todos valores de los objetivos de los individuos que se le asignan. Cuando este método se implementa sobre un sistema heterogéneo, puesto que todos los esclavos computan funciones objetivo idénticas, para un número idéntico de soluciones, los esclavo usualmente terminan su ejecución en un tiempo similar [89].

En el segundo método, las *k* funciones objetivos se mapean cada una a un esclavo. Puesto que la complejidad de las funciones puede variar, este método requiere una adecuada distribución de las *k* funciones a los distintos procesadores. El balance de carga puede ser de utilidad para distribuir adecuadamente el costo computacional entre los distintos procesadores utilizados, sin embargo, el esfuerzo para esto podría exceder cualquier ganancia final [89].

El último método distribuye el cálculo de la función objetivo en si misma, entre múltiples procesadores. Cada una de las funciones objetivo es particionada para su resolución entre múltiples procesadores. Este método es adecuado para problemas realmente grandes.

5.2.2. Modelo de Difusión

En busca de una mejora en el desempeño, otra alternativa para el desarrollo de algoritmos evolutivos paralelos es la paralelización de los distintos operadores evolutivos. Puesto que estos operadores son relativamente sencillos, en general, la relación entre el costo de la comunicación y el costo computacional hace que la aceleración que se pueda obtener, cuando existe, sea escasa. Sin embargo, el

modelo de difusión propone un esquema de selección distribuida de elementos pertenecientes a distintas subpoblaciones.

Como en el modelo de maestro-esclavo, en el modelo de difusión se considera conceptualmente una única población donde en cada procesador se tienen sólo unos pocos individuos. Este modelo supone que el cruzamiento entre ciertas subpoblaciones llevará a buenas soluciones, las cuales se encuentran en diferentes áreas, distribuidas convenientemente cuando se considera toda la población. Por tanto, este modelo precisa la imposición de alguna estructura que restrinja la selección y recombinación entre elementos que se encuentran en ciertas subpoblaciones. Debido a que la selección de individuos se realiza entre distintas subpoblaciones, el costo de la comunicación puede ser alto en este modelo.

5.2.3. Modelo de Islas

Cuando se utilizan sistemas de cómputo fuertemente acoplados, puesto que la relación del tiempo de cómputo respecto al tiempo de comunicación es alta, la paralelización del cálculo de la función objetivo o del procedimiento de selección son alternativas válidas para conseguir la aceleración de la ejecución. Sin embargo, en países como el Paraguay, este tipo de máquinas paralelas representa un costo prohibitivo para la mayor parte de las empresas e instituciones de enseñanza e investigación. Por lo tanto, la utilización de un conjunto de estaciones de trabajo como una máquina paralela (COW), ya sea conectadas a una red de área local (*Local Area Network - LAN*) o conectadas a través de Internet, se vuelve la única alternativa posible.

En estos entornos el costo de la comunicación es elevado, siendo deseable una granularidad mayor en el paralelismo. Además, es necesario considerar otras cuestiones, como la posible pérdida de una conexión, heterogeneidad de procesadores, etc. Una posibilidad de paralelización de fácil adaptación al entorno computacional señalado es la ejecución simultánea de varios MOEAs en procesadores

diferentes y que sus resultados se comparen, contrasten o combinen. A este esquema de paralelización se le denomina modelo de islas.

El modelo de islas está basado en el fenómeno natural de que las poblaciones se encuentran, usualmente, relativamente aisladas unas de otras. Los pMOEAs basados en este modelo se denominan MOEAs de multipoblación o distribuidos. La característica principal de los pMOEAs basados en el modelo de islas es que los individuos de una población particular pueden migrar de forma ocasional a alguna otra.

Conceptualmente, en el modelo de islas, una población se divide en un número de poblaciones separadas e independientes. Los diferentes operadores evolutivos trabajan en cada isla, lo que implica que las distintas poblaciones se encuentran buscando en regiones diferentes del espacio de búsqueda. Cada isla también podría tener distintos parámetros así como diferente estructura de MOEA. Además, individuos de una isla podrían migrar a otra isla de acuerdo a algún criterio.

Los distintos procesos que intervienen en la búsqueda de soluciones pueden comunicarse entre si utilizando distintas topologías de interconexión, tanto lógicas como físicas.

El modelo de islas requiere la selección de políticas de migración que señalen entre otras:

- la manera en que los individuos migrarán,
- el número de migrantes,
- la frecuencia de migración,
- de donde se seleccionarán los elementos a migrar y como se realizará el reemplazo de los elementos en una población por los migrantes provenientes de otras poblaciones,
- los distintos parámetros y algoritmos a utilizar en cada una de las diferentes islas.

Definiendo de forma conveniente la política de migración, este modelo puede aplicarse a varias arquitecturas paralelas, especialmente en sistemas paralelos de memoria distribuida.

Existen básicamente cuatro variantes de pMOEAs basados en el modelo de islas [89]:

1. homogéneos: en todas las islas se ejecutan el mismo MOEA con parámetros idénticos;
2. heterogéneos: en las distintas islas se ejecutan distintos MOEAs o el mismo MOEA con distintos parámetros;
3. cada isla evalúa un conjunto de funciones distinto;
4. en cada isla se representa una región diferente del dominio fenotípico o genotípico.

5.2.4. Esquema de paralelización utilizado

Para la implementación de MOEAs paralelos se precisan considerar diferentes cuestiones como:

1 - Determinar la plataforma paralela a utilizar.

2 - Determinar que modelo de paralelización utilizar.

3 - Determinar si se paralelizarán algoritmos existentes o si se justifica el desarrollo de un nuevo algoritmo desarrollado específicamente para la plataforma paralela.

En el primer punto, la disponibilidad de recursos ha condicionado desarrollar algoritmos paralelos pensados para ser implementados principalmente en COWs. A partir de esto, se ha seleccionado el modelo de islas debido a que es el que mejor se adecua a la arquitectura paralela disponible.

En cuanto al tercer punto, debido al carácter exploratorio del presente trabajo, se ha considerado importante identificar las características que poseen los algoritmos ya existentes y que podrían ser adecuadas reproducir en posteriores desarrollos. Por ello, se ha decidido desarrollar aplicaciones paralelas a partir de distintos MOEAs previamente desarrollados y utilizados exitosamente para resolver

una amplia gama de problemas. En particular, se ha decidido la utilización de los algoritmos presentados en el Capítulo 4. Estos representan distintas características de las técnicas de la computación evolutiva multiobjetivo y son fácilmente adaptables al modelo de paralelización seleccionado.

Determinados la plataforma de cómputo, los algoritmos a paralelizar y el modelo de paralelización, surgen nuevas consideraciones relativas al diseño de la propuesta de paralelización, en particular, con relación a la política de migración a utilizar. Primeramente, se precisa definir un criterio para la selección de soluciones a enviar . Una elección de sentido común indica la selección de los mejores individuos de una población en el momento que la migración ocurre: el conjunto de soluciones nodominadas. Puesto que el número de elementos en este conjunto puede ser considerablemente grande, es necesaria la utilización de un parámetro que limite el número máximo de soluciones a enviar a fin de evitar la sobrecarga en el proceso de envío y recepción de soluciones.

El Algoritmo 12 ilustra el procedimiento de selección de elementos a migrar utilizado en el presente trabajo. El procedimiento recibe como entrada una población $P(t)$ y el número máximo de elementos a seleccionar para la migración n_{mig}, proporcionando como salida una población de elementos a migrar P_{mig}. Primeramente se determina el conjunto $P_{known}(t)$ de elementos nodominados. Puesto que la mayor parte de MOEAs basan su funcionamiento en la identificación de los distintos frentes de elementos nodominados existentes en cada generación, determinar este conjunto podría requerir un mínimo de sobrecarga. Por ejemplo, en el caso del SPEA, se tiene que $P_{known}(t) = P'$. Si el número de soluciones nodominadas en $P(t)$ es inferior al máximo número de elementos a migrar, se seleccionan todos los elementos nodominados para formar parte de P_{mig}, en caso contrario se conforma P_{mig} seleccionando al azar de P_{known}, un total de n_{mig} soluciones distintas.

En cuanto a la frecuencia en que ocurrirá la migración existen varias opciones, entre ellas:

- enviar individuos transcurrido un número fijo de generaciones;
- enviar individuos de forma probabilística;

Algoritmo 12 Procedimiento de selección de elementos a migrar.

Procedimiento : mig_selec_procedure
Entrada : Una población $P(t)$
El número máximo de elementos a migrar n_{mig}
Salida : Una población de elementos a migrar $P_{mig}(t)$
Determinar el conjunto de elementos nodominados $P_{known}(t)$
Inicializar una estructura $P_{mig} = \emptyset$.
if $||PF_{known}||_c \leq n_{mig}$ **then**
$P_{mig}(t) = PF_{known}(t)$
else
while $||P_{mig}||_c \leq n_{mig}$ **do**
Seleccionar aleatoriamente un individuo *migrante* de $P_{known}(t)$ que no haya sido previamente incluido en P_{mig}
Agregar *migrante* a $P_{mig}(t)$
end while
end if

- enviar individuos de forma adaptativa.

En el primer caso, se determina el número de generaciones que deben transcurrir para que se produzca un envío. En el segundo, se proporciona como parámetro del algoritmo la probabilidad que un envío ocurra en una determinada generación. En ambos, se precisa de un parámetro que debe ser determinado de antemano. La tercera opción planteada, se basa en modificar la frecuencia de migración conforme el proceso de evolución ocurre, de suerte a realizar la búsqueda de soluciones de manera más eficiente. En el presente trabajo se utiliza un procedimiento de migración probabilístico a fin de sentar las bases para la exploración posterior de métodos de migración adaptativos.

Tanto el envío como la recepción de las soluciones puede realizarse básicamente de dos formas:

- síncrona;

- asíncrona.

En el caso síncrono, al momento en que la comunicación ocurre, el MOEA se bloquea en espera de soluciones provenientes de otras poblaciones o la confirmación de la recepción de las soluciones enviadas. En cambio cuando la comunicación es asíncrona, los MOEAs pueden continuar inmediatamente después que han ejecutado una primitiva de envío así como la verificación, sin espera, de la

recepción de soluciones. Además de la reducción en el tiempo de espera, la comunicación asíncrona es adecuada para lidiar con la posible pérdida de datos y elementos de procesamiento. Por ello, en este trabajo se ha determinado la utilización de primitivas de comunicación asíncronas, tanto para el envío como para la recepción de soluciones.

En cuanto a la forma en que procederá el reemplazo de elementos en una población, por los elementos recibidos, existen varias opciones, como:

- utilización de una población extra de intercambio;
- reemplazo aleatorio de elementos de una población por elementos recibidos;
- reemplazo aleatorio de elementos dominados por elementos recibidos.

La última de las opciones citadas fue utilizada en el presente trabajo, como se presenta en el Algoritmo 13. La primera opción fue desechada puesto que exige modificaciones mayores a los algoritmos utilizados. El reemplazo aleatorio de elementos de la población por elementos recibidos fue también desechado puesto que buenas soluciones podrían perderse al ser reemplazadas por soluciones migrantes inferiores. Sin embargo, la técnica de reemplazo aleatorio de elementos dominados por elementos recibidos, asegura que las soluciones nodominadas de la población actual con respecto al conjunto de migrantes sea preservado. Además, este método no asegura la eliminación de la peor solución. De este modo, a la vez de asegurar el mantenimiento de las buenas soluciones, posee una probabilidad mayor a cero que las características genéticas de distintos frentes sean conservadas.

Existen varias formas de implementar un procedimiento de reeplazo aleatorio de elementos dominados por elementos recibidos. Una de ellas es determinar para cada elemento de la población recibida $P_{recv}(t)[i]$ una lista con los elementos en la población $P(t)$ a los cuales domina y seleccionar de esta lista el elemento a reemplazar. La lista de elementos dominados se conforma recorriendo la población $P(t)$ y determinando los elementos que son dominados uno a la vez. Este procedimeinto es costoso en

Algoritmo 13 Procedimiento utilizado para el reemplazo de elementos.

Procedure: replacement_procedure
Entrada : Población recibida $P_{recv}(t)$
Población destino $P(t)$
Salida : Población $P(t)$ posiblemente modificada.
Desordenar aleatoriamente los indices de la población P para que los reemplazos sean sin prioridad
for $i = 1$ to $||P_{recv}(t)||_c$ **do**
  for $j = 1$ to $||P(t)||_c$ **do**
    if $P_{recv}(t)[i] \succ P(t)[j]$ **then**
      Reemplazar elemento $P(t)[j]$ por $P_{recv}(t)[i]$
      break
    end if
  end for
end for

términos computacionales puesto que para cada elemento en P_{recv} es necesario determinar la lista de nodominados para luego seleccionar un elemento de forma aleatoria. Como alternativa, el Algoritmo 13 propone primeramente desordenar la población aleatoriamente, luego en vez de formar toda la lista para seleccionar el elemento dominado, simplemente el primero encontrado es reemplazado. Para los algoritmos que no imponen un orden en los elementos de la población el procedimiento inicial puede eliminarse.

Finalmente, resta definir la topología de migración a utilizar. Durante el desarrollo del presente trabajo, se han explorado dos formas de paralelización de MOEAs:

- pMOEAs con topología de migración centralizada.
- pMOEAs con topología de migración descentralizada.

En el primer modelo, presentado en [91], las soluciones se intercambian de una isla a otra pasando primero a través de un proceso que se encarga de recibir las soluciones de las distintas islas y reenviarlas. Debido al potencial cuello de botella que limita su escalabilidad, el mismo fue desechado tras un periodo de experimentación con el mismo. En el modelo con migración descentralizada, cada decisión de enviar o recibir soluciones es llevado a cabo en cada una de las islas, eliminando el cuello de botella del modelo centralizado.

Algoritmo 14 Procedimiento colector (pColector).

Procedimiento: procedure_colector
Entrada : Número de pMOEAs p,
Tipo y parámetros de los MOEAs a inicializar: $< tipoMOEA_1, \text{parámetros}_{moea_1} > \ldots, < tipoMOEA_p, \text{parámetros}_{moea_p} >$
Número máximo de soluciones deseadas: $N_{colector}$
Salida : Un archivo con las soluciones nodominadas encontradas.
Inicializar población $P_{colector} = \emptyset$ para almacenar resultados provenientes de los pMOEAs
Agregar el colector a equipo de trabajo
Iniciar los p pMOEAs con sus parámetros correspondientes.
while $p > 0$ **do**
  Esperar resultados de los pMOEAs
  Recibir resultados en $P_{colector}$
  Eliminar elementos cubiertos en $P_{colector}$
  if $||P_{colector}||_c > N_{colector}$ **then**
    Aplicar el procedimiento de clustering del SPEA a $P_{colector}$ (Algoritmo 7)
  end if
  if Los resultados recibidos están marcados como finales **then**
    Reducir la cuenta de procesos MOEAs, $p = p - 1$
  end if
end while
Eliminar elementos cubiertos existentes en $P_{colector}$
Escribir los resultados obtenidos en un archivo

El modelo propuesto con topología de migración descentralizada se basa en la utilización de dos tipos distintos de procesos, formando un equipo de trabajo:

- un proceso colector; y
- n procesos MOEAs.

Estos procesos se pueden obtener implementando los Algoritmos 14 y 15. El colector se encarga de iniciar las distintas instancias de los MOEAs utilizados y de almacenar soluciones obtenidas por éstos. A los distintos MOEAs que realizan la búsqueda de soluciones se les ha agregado un procedimiento de migración aleatorio y uno de recepción asíncrona de soluciones.

El proceso colector primeramente lee el número de procesos MOEA que trabajaran en la búsqueda de soluciones. Puesto que cada uno de éstos puede corresponder a un tipo de MOEA distinto o bien al mismo tipo de algoritmo evolutivo pero con diferentes parámetros, el colector también precisa

información sobre el tipo de MOEA que se utilizará así como los parámetros que le corresponden, para inicializarlos posteriormente en forma conveniente. Así mismo, el colector recibe la especificación del número de soluciones deseadas, denotado por $N_{colector}$. Seguidamente, el colector crea las estructuras para almacenar los resultados provenientes de los distintos procesos MOEA (población $P_{colector}$). Luego, el colector se agrega a un grupo de trabajo e inicia cada uno de los procesos MOEA con sus parámetros específicos. Esto puede ser implementado utilizando primitivas de comunicación de grupo proveídas por librerías de paso de mensajes (ej. PVM), lo que facilita la comunicación entre los distintos procesos utilizados.

Como se explica más adelante, durante la evolución, cada uno de los MOEAs que interviene en la búsqueda envía a todos los elementos que componen el equipo de trabajo un porcentaje de las mejores soluciones obtenidas. Cuando el colector recibe estas soluciones las almacena en $P_{colector}$. A fin de mantener en el colector sólo las mejores soluciones, se eliminan de $P_{colector}$ las soluciones cubiertas. En caso que el número de elementos en el conjunto de soluciones del colector supere un máximo de soluciones deseadas, se procede a la reducir el tamaño del conjunto utilizando el procedimiento de clustering del SPEA (Algoritmo 7). Si los resultados recibidos corresponden a los de la última generación de un proceso MOEA, se reduce la cuenta de MOEAs en ejecución. Cuando ésta cuenta es igual a cero, se procede a eliminar las soluciones cubiertas existentes en $P_{collector}$ y la escritura (salida final) de los resultados obtenidos.

El Algoritmo 15 presenta el marco general propuesto para la implementación de los distintos pMOEAs. Al comienzo, se realizan algunas tareas propias de inicialización. Primeramente, se reciben desde el colector los distintos parámetros del MOEA utilizado. Además de los parámetros usuales, probabilidad de cruzamiento, selección, mutación, radio de nicho, etc., también se reciben la probabilidad de migración (p_{mig}) así como el número máximo de elementos nodominados a migrar (n_{mig}). Tras la fase de recepción de los parámetros del algoritmo, cada pMOEA se agrega a su equipo de

Algoritmo 15 Algoritmo genético paralelo general.

Procedure: procedure_pMOEA
Recibir parámetros usuales del MOEA, así como la probabilidad de migración p_{mig} y el número de elementos nodominados a migrar n_{mig}.
Unirse al equipo de trabajo
Generar la población inicial $P(0)$ de forma aleatoria y hacer $t = 0$
repeat
  if se han recibido soluciones provenientes de otros procesos **then**
    Almacenar las soluciones recibidas en $P_{recv}(t)$
    replacement_procedure $(P(t), P_{recv})$ `//Algoritmo 13`
  end if
  Generar nueva población $P(t+1)$
  $t = t+1$
  if $rand(0,1) < p_{mig}$ **then**
    $P_{mig} =$ **mig_select__procedure**$(P(t))$ `//Algoritmo 12`
    Enviar P_{mig} a todos los elementos en el equipo de trabajo
  end if
until Alcanzar el criterio de parada
Enviar $P_{known}(t)$ al colector con señal de finalización
Salir del grupo de trabajo y terminar ejecución

trabajo.

Luego de la inicialización, los pMOEAs realizan el proceso evolutivo. En cada generación, se determina si se han recibido elementos provenientes de otras subpoblaciones. Si este es el caso, se realiza la recepción de los mismos en P_{recv} (población recibida). Luego de la recepción de los elementos de P_{recv}, se procede al reemplazo de $||P_{recv}||_c$ elementos de P dominados por elementos de P_{recv}. En el peor caso, ninguna solución en P es dominada por alguna solución en P_{recv}. Cuando esto ocurre, el procedimiento de reemplazo no tiene ningún efecto sobre la estructura de la población genética, aunque si un efecto negativo sobre la velocidad del algoritmo. Posteriormente, se realizan los procedimientos usuales de evolución para generar una nueva población. Luego, se determina si corresponde la migración de elementos en esa generación t. Si corresponde la migración, se obtienen los elementos nodominados existentes en $P(t)$ para el caso de los algoritmos sin población externa y luego se selecciona de entre ellos, aleatoriamente, como máximo n_{mig} individuos, los cuales son enviados a todos los procesos que componen el equipo de trabajo. Transcurrido un cierto número de generaciones o al-

guna otra condición, se envían todos los elementos de $P_{known}(t)$ al procedimiento colector. Al alcanzar un número máximo de generaciones, el pMOEA envía todas sus soluciones al colector y finaliza.

El modelo propuesto garantiza que recibida una solución, en una de las islas, ésta será aceptada para formar parte de la población sólo si domina a alguna solución en el conjunto. Note que no se utiliza ningún procedimiento de almacenamiento o control sobre el conjunto de soluciones que ya han sido enviadas. Si bien tal procedimiento podría ser beneficioso para evitar el envío repetitivo de soluciones ya encontradas por un procesador, esto adhiere complejidad adicional al pMOEA en la etapa de envío que podría no ser correspondido por una mejora en la eficiencia del algoritmo. Además, puesto que es de esperar que los envíos se produzcan luego de cierto número de generaciones y que el conjunto de soluciones nodominadas cambie y por tanto se modifique el conjunto de soluciones a enviar, al menos parcialmente. Por otro lado, las soluciones a migrar son seleccionadas de forma aleatoria entre el conjunto de soluciones nodominadas y enviadas de forma asíncrona, por lo que aún cuando por azar se repita el envío de soluciones, la recepción de ellas no está garantizado. Es más, aunque se garantice la recepción, no se garantiza que el receptor acepte la solución pues podría tratarse de una solución dominada con respecto a la población del receptor.

Capítulo 6

Comparación Experimental de pMOEAs

Para la mayoría de los problemas de optimización multiobjetivo, el conocimiento del frente Pareto óptimo ayuda al tomador de decisiones a seleccionar aquella solución que representa el mejor compromiso. Generar el frente Pareto-óptimo puede ser computacionalmente costoso o incluso imposible, en especial en problemas reales de ingeniería. Entonces, lo único que se puede pretender es obtener una buena aproximación al frente Pareto óptimo verdadero. Generalmente, las características del espacio de búsqueda de los problemas de optimización multiobjetivo del mundo real hacen que sea imposible o inadecuada la utilización de métodos exactos para la obtención de esta aproximación. Como se ha dicho, los algoritmos evolutivos multiobjetivo son una alternativa práctica en la búsqueda de soluciones de compromiso para problemas reales donde los métodos exactos son inaplicables o ineficientes.

Existen varios MOEAs que pueden ser utilizados para aproximar las soluciones de un problema. El interés está en identificar qué algoritmo consigue el mejor desempeño para el problema considerado. Para develar las ventajas y desventajas de los diferentes algoritmos, el enfoque tradicional de la comunidad de científicos e ingenieros que trabajan en EMOO ha sido la comparación experimental de los distintos algoritmos [2, 3, 86, 87, 97, 98, 99, 103]. Mientras que, los esfuerzos teóricos son de una

cantidad considerablemente menor [60, 71].

La comparación experimental del desempeño de los algoritmos es indirecta en el sentido que se comparan implementaciones particulares de los algoritmos, no los algoritmos propiamente dichos. En este trabajo, por simplicidad, se hará referencia a los resultados obtenidos por la ejecución de una implementación particular de un algoritmo como el resultado del algoritmo.

La noción de desempeño incluye tanto la calidad del resultado, como los recursos computacionales necesarios para obtenerlo. Sin embargo, puesto que la comparación es indirecta, generalmente se considera sólo la calidad del resultado, siendo el costo computacional una cuestión secundaria. Como un ejemplo, puesto que la velocidad de cómputo es una cuestión secundaria, se comparan resultados obtenidos tanto por algoritmos implementados en Matlab como algoritmos implementados en C [86]. Entonces, a fin de realizar las comparaciones, se utiliza el número de evaluaciones de la función objetivo o bien el número de generaciones como una constante, para comparar distintos MOEAs sin tener en cuenta el tiempo que lleva realizar una corrida o los recursos de memoria necesarios.

En optimización multiobjetivo no existe un criterio único que permita establecer con facilidad si un conjunto de aproximación es mejor que otro puesto que se pueden considerar varias medidas de calidad. Por ejemplo: la cercanía al conjunto Pareto óptimo, el número de soluciones obtenidas, la distribución de las soluciones, representan métricas alternativas [86].

Para comparar los resultados obtenidos por distintos algoritmos evolutivos se han desarrollado varias métricas de desempeño, las cuales buscan capturar las características que hacen a una aproximación del conjunto Pareto-óptimo mejor que otra en algún criterio. Por lo general, éstas asignan a un conjunto aproximación particular un número que refleja un aspecto de calidad determinado [36, 86]. Otras métricas utilizadas para determinar si un conjunto aproximación es mejor que otro son binarias y asignan a un par de conjuntos a comparar un número que indica como éstos se relacionan [103]. En general, no existe un acuerdo común sobre cuáles medidas de desempeño debería utilizarse, por lo

que una práctica usual es utilizar una combinación de métricas a fin de comparar las aproximaciones obtenidas en diferentes aspectos. Así, la comparación en sí misma se vuelve multiobjetivo.

A fin de determinar la mejora en el desempeño que se puede obtener con la utilización de pMOEAs y considerando la falta de estudios experimentales que analicen las distintas cuestiones relativas al desarrollo de los mismos y su desempeño con relación a los algoritmos secuenciales, en este trabajo se han paralelizado y comparado seis algoritmos evolutivos siguiendo el modelo propuesto en la sección 5. Estos algoritmos son: el MOGA [42, 43], el NPGA [54, 55], el NSGA [81, 82, 83], el SPEA [102, 97, 103], el NSGA-II [33], y el CNSGA-II [34]. Los mismos fueron seleccionados considerando su importancia de acuerdo al número de variaciones e implementaciones de los mismos presentados en [100] así como su utilización en trabajos previos de comparación [68, 90, 96]. Los resultados obtenidos se han comparado utilizando distintas métricas experimentales [36, 86, 97]. Primeramente se ha utilizado un conjunto de funciones de prueba [99] y luego un problema real de optimización [80, 92].

En el presente capítulo, se presentan las distintas métricas utilizadas en la comparación y las funciones de prueba seleccionadas. Así mismo, se presenta el análisis de los resultados obtenidos por los distintos algoritmos seleccionados a la luz de las distintas métricas y sobre las funciones de prueba consideradas. Por su parte, en el próximo capítulo, se presenta un problema real de ingeniería y el correspondiente análisis de los resultados obtenidos.

6.1. Métricas de Desempeño

6.1.1. Generación Total de Vectores Nodominados

La métrica de generación de vectores nodominados (*Overall Non-dominated Vector Generation* - ONVG) se utiliza para medir el número de soluciones en el frente Pareto calculado por un MOEA

al final de una corrida (PF_{known}) y se define como [86]:

$$\text{ONVG} = ||PF_{known}||_c \tag{6.1}$$

Aunque la cuenta de soluciones obtenidas proporciona cierta información sobre la efectividad de un MOEA, no dice nada acerca de que tan lejos se encuentra de los vectores existentes en PF_{true}. Además, muy pocos vectores en PF_{known} podrían no ayudar suficientemente al tomador de decisiones, mientras que demasiadas podrían no ser tampoco de utilidad, por lo que es difícil determinar que valores para ONVG sería adecuado. En general, se prefiere conjuntos con el mayor valor posible de ONVG, lo que será adoptado también en este trabajo.

6.1.2. Razón de la Generación Total de Vectores Nodominados

De acuerdo al problema considerado, la cardinalidad del conjunto de soluciones propuestas por un algoritmo (PF_{known}) puede variar considerablemente. Por tanto, podría resultar de mayor utilidad indicar la razón entre la cantidad de vectores nodominados encontrados por cada algoritmo y la cantidad de vectores nodominados existentes en PF_{true}, por lo que se define la métrica razón de la generación total de vectores nodominados (*Overall Non-dominated Vector Generation Ratio* - ONVGR) como [86]:

$$\text{ONVGR} = \frac{||PF_{known}||_c}{||PF_{true}||_c} \tag{6.2}$$

Un valor de 1 indica que el MOEA calcula el mismo número de soluciones nodominadas que existe en PF_{true}.

6.1.3. Número de soluciones nodominadas verdaderas

Aunque las métricas anteriores sean de utilidad, no proporcionan una idea muy clara sobre el desempeño real del algoritmo en la búsqueda, ya que el mismo puede dar como resultado un conjunto con el

mismo número de elementos que el frente Pareto óptimo real, pero sin que exista una correspondencia entre éstos y los elementos hallados. Consecuentemente, en [36] se define una métrica que sólo cuenta la cantidad de vectores nodominados propuestos que en efecto pertenecen al frente Pareto óptimo real (N) de la siguiente manera [36]:

$$\mathrm{N} = \{\, x \mid x \in PF_{known} \wedge x \in PF_{true} \,\} \tag{6.3}$$

6.1.4. Razón del error

La métrica razón del error (*Error Ratio*), o simplemente Error, reporta la proporción de vectores objetivo en PF_{known} que no son miembros de PF_{true}. La definición matemática es [86]:

$$\mathrm{E} = \frac{\sum_{i=1}^{\mathsf{n}} e_i}{\mathrm{ONVG}} \tag{6.4}$$

donde n, es el número de vectores en PF_{known} y

$$e_i = \begin{cases} 0 \text{ si el vector } i, \ (i = 1, \ldots, \mathsf{n}) \text{ pertenece a } PF_{true} \\ \\ 1 \text{ en otro caso} \end{cases} \tag{6.5}$$

Por ejemplo, un valor $\mathrm{E} = 0$ señala que cada vector obtenido por el MOEA en PF_{known} está en PF_{true}; $\mathrm{E} = 1$ indica que ningún vector obtenido por el MOEA es en realidad una solución preteneciente a PF_{true}.

6.1.5. Distancia Generacional

La métrica distancia generacional (*Generational Distance*) representa que tan lejos se encuentra PF_{known} de PF_{true} y se define como [86]:

$$\mathrm{GD} = \frac{(\sum_{i=1}^{\mathsf{n}} d_i^{min^2})^{\frac{1}{2}}}{\mathsf{n}} \tag{6.6}$$

donde d_i^{min} es la distancia Euclidiana (en el espacio objetivo) entre cada vector objetivo en PF_{known} y su correspondiente más cercano en PF_{true}. Un resultado de 0 indica que $PF_{true} = PF_{known}$, mientras que otro valor indica que PF_{known} se desvía de PF_{true}.

6.1.6. Error Máximo del Frente Pareto

Cuando se compara PF_{known} con PF_{true}, se desea determinar que tan separados están ambos conjuntos y si se parecen en su forma. La métrica error máximo del frente Pareto (*Maximum Pareto Front Error* - ME) se define con el propósito de determinar una banda de error máxima cuando se considera PF_{known} con respecto a PF_{true}. Para ello, para cada elemento $PF_{known}[i]$, se obtiene su correspondiente valor de d_i^{min}, obteniéndose luego el mayor de estos valores [86].

$$\text{ME} = max(\{d_1^{min}, d_2^{min}, \ldots, d_{\mathsf{n}}^{min}\}) \tag{6.7}$$

Un resultado igual a 0 señala que $PF_{known} \subseteq PF_{true}$ mientras que otro valor indica que al menos un vector en PF_{known} no está en PF_{true}.

6.1.7. Espaciamiento

A fin de medir la distribución de las soluciones en PF_{known} se define la métrica espaciamiento (*Spacing* - *S*) como [86]:

$$S = \sqrt{\frac{1}{n-1}\sum_{i=1}^{\mathsf{n}}(\overline{d^{min}} - d_i^{min})^2} \tag{6.8}$$

Siendo

$$\overline{d^{min}} = \frac{\sum_{i=1}^{\mathsf{n}} d_i^{min}}{\mathsf{n}} \tag{6.9}$$

Un valor de cero significa que todos los miembros de PF_{known} están igualmente espaciados.

6.1.8. Cobertura

Si bien las métricas presentadas proporcionan una idea acerca del desempeño de un MOEA para la búsqueda de soluciones, éstas no realizan una comparación directa de las soluciones obtenidas por un conjunto solución con respecto a otro. Además, para el cálculo se requiere el conocimiento del PF_{true} o la utilización de una aproximación al mismo. En [99] se define la métrica cobertura que representa la relación del número de vectores en el espacio objetivo, encontrados por un dado algoritmo, que son mejores que los encontrados por otro. Sea PF_{known} el conjunto de soluciones propuesto por un algoritmo y PF'_{known} el conjunto de soluciones propuesto por otro, se define entonces la métrica cobertura (*Coverage - C*) como una función que mapea el par ordenado $(PF_{known}, PF'_{known})$ al intervalo $[0,1]$:

$$C(PF_{known}, PF'_{known}) = \frac{||\{PF'_{known}[i] \mid \exists PF_{known}[j] \, , \; PF_{known}[i] \succeq PF'_{known}[j]\}||_c}{||PF'_{known}||_c} \tag{6.10}$$

Un valor de 1 indica que todas las soluciones en PF'_{known} son dominadas por las soluciones existentes en PF_{known}, mientras un valor de cero indica que ninguna es dominada. El interés está en determinar cuál algoritmo es en promedio el que proporciona el menor número de soluciones cubiertas y el mayor número de soluciones que cubren a las soluciones obtenidas por el otro.

6.2. Funciones de prueba ZDT

En [28] se identifican diferentes características que pueden hacer que los MOEAs tengan dificultades para converger al conjunto Pareto óptimo y mantener la diversidad en la población. La multimodalidad, la decepción y los óptimos aislados son áreas problemáticas bien conocidas que dificultan la convergencia de los EAs cuando se utilizan para optimización monobjetivo [31]. El mantenimiento de la diversidad en la población es necesaria para la obtención de un frente Pareto bien distribuido. De acuerdo al problema considerado, ciertas características propias del conjunto Pareto óptimo corres-

pondiente pueden dificultar que el MOEA encuentre una variedad de soluciones Pareto-óptimas bien distribuidas:

- convexidad o no-convexidad,
- discretitud,
- no-uniformidad.

En [99] se desarrolló un conjunto de funciones de prueba que contempla las distintas posibilidades existentes. Estas funciones consideran la minimización de dos objetivos y cada una de ellas está estructurada de la misma forma y consisten de tres funciones f_1, g y h:

$$\begin{aligned} &\text{Minimizar } F(\mathbf{x}) = (f_1(x_1), f_2(\mathbf{x})) \\ &\quad \text{sujeto a } f_2(\mathbf{x}) = g(x_2,\ldots,x_m)h(f_1(x_1), g(x_2,\ldots,x_m)) \\ &\quad\quad \text{donde } \mathbf{x} = (x_1,\ldots,x_m) \end{aligned} \tag{6.11}$$

La función f_1 es una función que depende únicamente de la primera variable de decisión, g es una función de las $m-1$ variables de decisión restantes y los parámetros de h son los valores de las funciones f_1 y g. Las funciones de prueba difieren en estas tres funciones así como en el número de variables m y en los valores que éstas pueden tomar. En el presente trabajo se utilizan cuatro de éstas funciones, a fin de comparar los distintos algoritmos con respecto a la convexidad, la no-convexidad, la discretitud y la multimodalidad. Las funciones utilizadas se definen formalmente como sigue.

Definición 6.1. Se definen cuatro funciones de prueba ZDT1, ..., ZDT4 que siguen el esquema de la ecuación 6.11:

- La función de prueba ZDT1 tiene un frente Pareto-óptimo convexo:

$$\begin{aligned} f_1(x_1) &= x_1 \\ g(x_2,\ldots,x_m) &= 1+9\cdot\sum_{i=2}^{m}\frac{x_i}{(m-1)} \\ h(f_1,g) &= 1-\sqrt{\frac{f_1}{g}} \end{aligned} \tag{6.12}$$

  Donde $m = 30$, y $x_i \in [0,1]$. El conjunto Pareto-óptimo se forma con $g(\mathbf{x}) = 1$.

- La función de prueba ZDT2 es la contraparte no-convexa de ZDT1:

$$\begin{aligned} f_1(x_1) &= x_1 \\ g(x_2,\ldots,x_m) &= 1+9\cdot\sum_{i=2}^{m}\frac{x_i}{(m-1)} \\ h(f_1,g) &= 1-(\frac{f_1}{g})^2 \end{aligned} \tag{6.13}$$

  Donde $m = 30$, y $x_i \in [0,1]$. El conjunto Pareto-óptimo se forma con $g(\mathbf{x}) = 1$.

- La función de prueba ZDT3 representa la característica de discretitud, su frente Pareto-óptimo consiste de varias partes convexas no contiguas:

$$\begin{aligned} f_1(x_1) &= x_1 \\ g(x_2,\ldots,x_m) &= 1+9\cdot\sum_{i=2}^{m}\frac{x_i}{(m-1)} \\ h(f_1,g) &= 1-\sqrt{\frac{f_1}{g}}-\frac{f_1}{g}\cdot\sin(10\cdot\pi\cdot f_1) \end{aligned} \tag{6.14}$$

  Donde $m = 30$, y $x_i \in [0,1]$. El conjunto Pareto-óptimo se forma con $g(\mathbf{x}) = 1$. La utilización de la función seno en h causa la discontinuidad en el frente Pareto-óptimo. Sin embargo, no existe discontinuidad en el espacio de parámetros.

- La función de prueba ZDT4 contiene 21^9 frentes Pareto-óptimos locales y, por tanto, prueba a

los algoritmos evolutivos con relación a su capacidad de lidiar con multimodalidad:

$$\begin{aligned} f_1(x_1) &= x_1 \\ g(x_2,\ldots,x_m) &= 1 + 10\cdot(m-1) + \sum_{i=2}^{m}(x_i^2 - 10\cdot\cos(4\cdot\pi\cdot x_i)) \\ h(f_1,g) &= 1 - \sqrt{\frac{f_1}{g}} \end{aligned} \tag{6.15}$$

Donde $m = 10$, $x_1 \in [0,1]$ y $x_2,\ldots,x_m \in [-5,5]$. El conjunto Pareto-óptimo global se forma con $g(\mathbf{x}) = 1$ y el mejor frente Pareto-óptimo local con $g(\mathbf{x}) = 1{,}25$. Note que no todos los conjuntos Pareto-óptimos locales son distinguibles en el espacio objetivo.

6.3. Descripción de los experimentos sobre funciones ZDT

Los seis algoritmos elegidos para este trabajo fueron implementados, para la resolución de las funciones de prueba presentadas en la Definición 6.1, de forma secuencial y paralela. Para las implementaciones secuenciales, con excepción del MOGA, los algoritmos seleccionados se implementaron de acuerdo a la literatura original de referencia. En el caso del MOGA, a diferencia del trabajo original, no se utilizó restricción de apareamiento de forma semejante a [99] e igualmente, se nombró al mismo como FFGA (*Fonseca y Flemming Genetic Algorithm*). Las implementaciones paralelas se realizaron utilizando el marco general propuesto en la sección 5.2.4. Fuera de las modificaciones pertinentes para adaptarlos al modelo de paralelización, no se realizaron otras variaciones con respecto a las versiones secuenciales de los algoritmos. En todos los casos los operadores de mutación y cruzamiento utilizados fueron mutación de un sólo bit y cruzamiento de un sólo punto.

Las soluciones de las distintas funciones de prueba se codificaron como cadenas binarias. Para decodificar las soluciones, se implementó la ecuación 2.1.

Todos los programas se escribieron en lenguaje C. Para las implementaciones paralelas se utilizaron las primitivas de comunicación proveídas por PVM v3.4.4 [39]. Los programas se compilaron

Característica	Descripción
Tipo de computadora	COW
Tipo de CPU	AMD K6-2 700MHz
Número de CPUs	20
Memoria	128 MB
Sistema Operativo	Red Hat Linux v7.3
Red de comunicación	Ethernet 100 Mb
Librería de comunicación	PVM 3.4.4

Tabla 6.1. Características del entorno computacional paralelo utilizado.

utilizando gcc -v2.96 para LINUX [1].

A fin de analizar el desempeño de las implementaciones secuenciales y paralelas de los MOEAs utilizados en este trabajo, para cada problema considerado se realizaron varias corridas distintas de los mismos. Las ejecuciones paralelas se realizaron utilizando una máquina PVM cuyas características principales se presentan en la Tabla 6.1. La asignación de procesos a cada CPU que compone el entorno paralelo utilizado lo realiza PVM. Las ejecuciones secuenciales se realizaron utilizando máquinas individuales que componen el COW presentado en la Tabla 6.1.

Aunque la combinación de algoritmos con el modelo de paralelización propuesto es de fácil implementación, en ninguna de las corridas realizadas se combinaron procesos MOEAs distintos. Esto es, no se consideraron equipos de trabajo compuestos de, por ejemplo, un proceso SPEA y uno NSGA.

Para cada problema considerado se realizaron 10 ejecuciones de cada implementación secuencial de los 6 algoritmos utilizados, y 10 ejecuciones de las versiones paralelas utilizando 1, 2 y 4 procesos pMOEAs. Esto es un total de 240 corridas (6 algoritmos x 4 implementaciones x 10 corridas) para cada problema de prueba. Para las diferentes corridas, se han utilizado diferentes semillas para la generación de números aleatorios. Además, cuando se utiliza en una ejecución más de un proceso pMOEA, estos también utilizan semillas diferentes, explorando así inicialmente en un subespacio mayor del espacio de búsqueda. En las distintas corridas paralelas se utilizó una probabilidad de migración (p_{mig}) igual a 0,5. En éstas, el número máximo de soluciones nodominadas que intercambian los distintos procesos

[1] El código puede ser obtenido por petición a *cvlucken@cnc.una.py*

Algoritmo	Parámetro						
	p_m	p_c	N	N'	σ_{share}	t_{dom}	t_{red}
CNSGA-II	0,01	0,8	100	-	-	-	0,7
NSGA-II	0,01	0,8	100	-	-	-	-
SPEA	0,01	0,8	100	100	-	-	-
NSGA	0,01	0,8	100	-	0,41	-	-
NPGA	0,01	0,8	100	-	0,41	10%	-
FFGA	0,01	0,8	100	-	0,41	-	-

Tabla 6.2. Parámetros utilizados en las diferentes corridas.

(n_{mig}) es 10.

En todos los casos se utilizaron los siguientes parámetros, según corresponda a cada MOEA utilizado (Tabla 6.2) :

- Tamaño de la población genética: 100.
- Tamaño de la población externa: 100.
- Probabilidad de cruzamiento: 0,8.
- Probabilidad de mutación: 0,01.
- Radio de nicho (σ_{share}): 0,41
- Presión de dominancia: 10.
- Tasa de reducción: 0,7.

Los resultados de las distintas ejecuciones se agruparon, obteniéndose un conjunto de soluciones nodominadas para cada algoritmo implementado correspondiente a la corrida. Por simplicidad, a la corrida secuencial se nombró como corrida a, mientras que, a las corridas paralelas que utilizan 1, 2 y 4 procesos pMOEA como corrida b, c y d respectivamente. Al conjunto de soluciones propuesto por la corrida a (secuencial) que utiliza al SPEA como algoritmo evolutivo se nombró como SPEA-a, mientras que el conjunto obtenido utilizando el NSGA como NSGA-a, y así para cada algoritmo

Corrida	Número de ejecuciones	p_{mig}	n_{mig}	Número de procesos	Usa colector
a	10	-	-	1	No
b	10	0.5	10	1	Si
c	10	0.5	10	2	
d	10	0.5	10	4	

Tabla 6.3. Características las distintas corridas.

utilizado. Del mismo modo, al conjunto de soluciones propuesto por la corrida paralela que utiliza un colector y un proceso pMOEA que utiliza como algoritmo de búsqueda al SPEA se nombró como SPEA-b y los resultados de las corridas con dos y cuatro procesos se nombraron como SPEA-c y SPEA-d, y de forma similar para los demás algoritmos. La Tabla 6.3 presenta en formato tabular los parámetros utilizados en las distintas corridas.

Para el cálculo de las métricas que requieren el conocimiento del frente Pareto óptimo real, se utilizó como aproximación el conjunto de soluciones nodominadas con respecto al conjunto unión de los resultados obtenidos considerando todas las ejecuciones realizadas.

6.4. Resultados experimentales sobre las funciones de prueba

6.4.1. Resultados de los experimentos sobre la función ZDT1

Los resultados de las métricas, utilizadas en este trabajo, para comparar los conjuntos solución calculados por las distintas implementaciones, sobre el problema de prueba ZDT1, se presentan en formato tabular al final de la presente sección.

La Tabla 6.6, de la página 113, presenta los valores de las métricas ONVG, ONVGR y N, sobre el conjunto de resultados obtenidos por los distintos pMOEAs para el problema de prueba ZDT1 en todas las corridas realizadas. Mientras, la Tabla 6.7 presenta los resultados de las métricas Error, GD, ME y Spacing. En la Tabla 6.8 se presentan los tiempos de ejecución promedio, medidos en segundos, para las ejecuciones llevadas a cabo en cada corrida. En estas tablas, los valores de las

métricas correspondientes a los distintos conjuntos se presentan ordenados de mejor a peor. En caso de empate, los conjuntos se consideran en la misma posición aunque, para facilitar la presentación de los resultados pueden figurar en posiciones distintas.

En la Tabla 6.9 se presentan los resultados de la métrica cobertura aplicada a los distintos conjuntos obtenidos por las implementaciones de los MOEAs de segunda generación ya que estos obtuvieron mejores resultados para las métricas de la tabla 6.6.

Por otra parte, las tablas 6.10, 6.11, 6.12 y 6.13, presentan los resultados de la aplicación de la métrica cobertura entre los conjuntos propuestos en las corridas a, b, c y d para cada algoritmo implementado, tanto de primera como de segunda generación.

A partir de los resultados presentados en las distintas tablas, se obtienen las siguientes conclusiones parciales:

- La Figura 6.1 ilustra los distintos valores de la métrica ONVG para cada conjunto solución considerado. Como puede verse, para cada algoritmo implementado, conforme aumenta el número de elementos de proceso utilizados para la obtención del conjunto solución, aumenta el valor de esta métrica.

- Para los conjuntos solución propuestos, considerando la métrica ONVG, existe una clara diferencia entre los conjuntos obtenidos por las implementaciones de los algoritmos que utilizan elitismo durante el proceso de evolución y aquellos que no, favoreciendo a los primeros.

- Para la métrica ONVG, el conjunto de soluciones NSGA-II-d es el mejor de todos, seguido por el NSGA-II-c, y en tercer lugar el SPEA-d. Las posiciones 4 a la 6 en la clasificación las ocupan los conjuntos CNSGA-II-d, CNSGA-II-c y SPEA-c, respectivamente.

- Si bien el conjunto de soluciones proveído por la implementación paralela que utiliza cuatro procesos SPEA supera al que utiliza cuatro procesos CNSGA-II, la relación se invierte en las

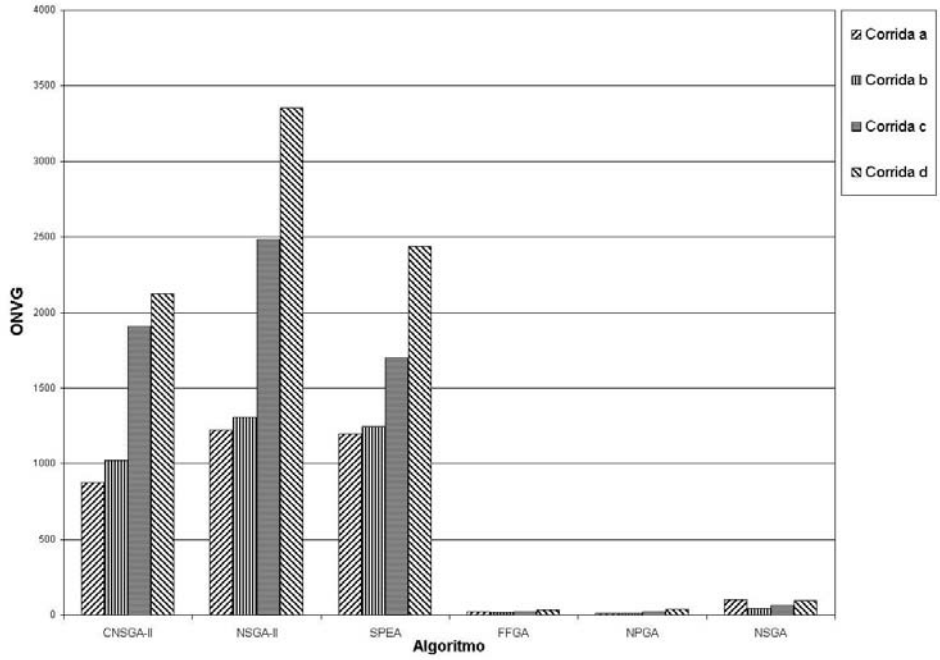

Figura 6.1. ZDT1: Resultado de la métrica ONVG para los distintos conjuntos.

corridas donde se utilizan dos procesos MOEA.

- Entre los conjuntos solución obtenidos, utilizando las implementaciones de algoritmos de primera generación considerados, son los propuestos por las implementaciones basadas en NSGA los que consiguen mejores resultados para la métrica ONVG. De ellos, para esta métrica, el mejor es el conjunto NSGA-a.

- Considerando la métrica ONVGR, los conjuntos se mantienen en el mismo orden de clasificación que para la métrica anteriormente considerada. Puesto que, finalmente, sólo representa una normalización del ONVG con respecto al número de soluciones consideradas en PF_{true}.

- La Figura 6.2 ilustra los valores de la métrica N para cada conjunto solución propuesto. Como en la Figura 6.1, se pueden apreciar las grandes diferencias existentes entre los resultados obtenidos por las implementaciones de los algoritmos de primera y segunda generación.

- Es importante notar el efecto del elitismo, ya que los algoritmos de primera generación son

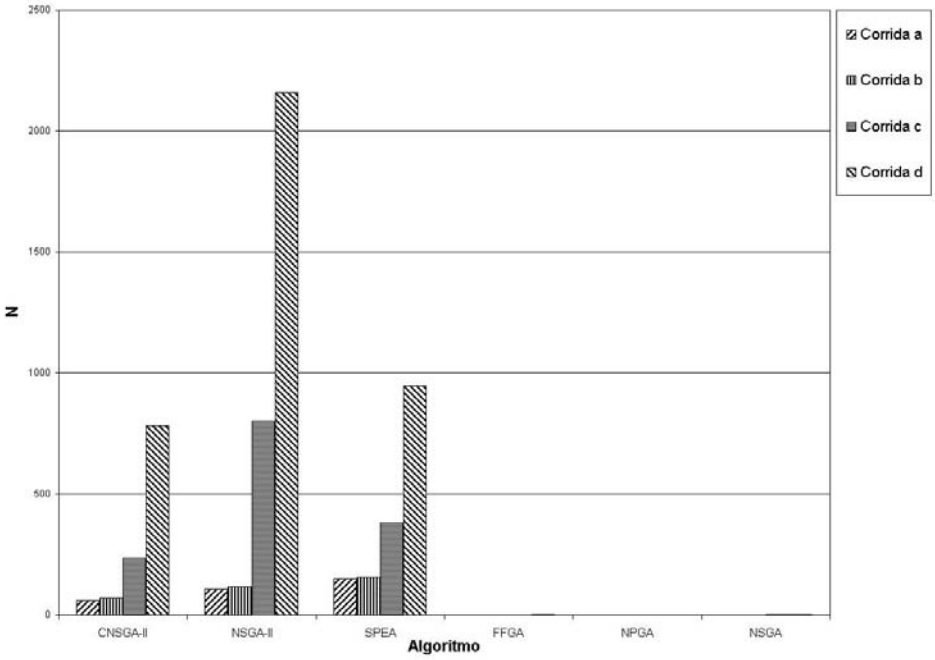

Figura 6.2. ZDT1: Resultado de la métrica N para los distintos conjuntos.

capaces de obtener soluciones en PF_{true} sólo cuando éstas son enviadas al proceso colector que las almacena evitando que sean eliminadas durante el proceso evolutivo. Para este problema, sin embargo, este tipo de elitismo es

- Para la métrica N, el mejor conjunto solución obtenido por implementaciones secuenciales es la del SPEA. Sin embargo, entre las implementaciones que utilizan dos y cuatro procesadores los mejores conjuntos fueron calculados utilizando el NSGA-II.

- La Tabla 6.4 presenta el radio de N para los conjuntos propuestos por las corridas paralelas con relación al propuesto por la corrida secuencial para cada algoritmo de segunda generación utilizado. Como puede verse, apenas existe variación en el número de soluciones propuestas por las corridas a y b de los distintos algoritmos presentados. Sin embargo, cuando se utilizan dos o más elementos de proceso, el número de soluciones propuestas crece considerablemente con el número de procesadores.

Relación	CNSGA-II	NSGA-II	SPEA
b/a	1.152542	1.084112	1.040268
c/a	3.966102	7.495327	2.563758
d/a	13.25424	20.16822	6.348993

Tabla 6.4. ZDT1: Razón de N entre los conjuntos obtenidos por las corridas paralelas y la secuencial.

- La Tabla 6.5 muestra la ganancia porcentual considerando el valor de N para los distintos conjuntos propuestos como solución al problema ZDT1 por las corridas de las implementaciones paralelas y la secuencial. La ganancia porcentual debida a la recepción de soluciones nodominadas enviadas por los distintos procesos, es del orden siguiente: SPEA 4%, NSGA-II 8% y CNSGA-II 15%. Note que el SPEA es el menos beneficiado.

- La ganancia porcentual en el número de soluciones obtenidas considerando las corridas c con relación a las corridas a es del orden del 156% para el SPEA, 649% para el NSGA-II y 296 para el CNSGA-II. Al agregar dos procesadores más, el crecimiento relativo de la corrida d con respecto a la a es del 1916% para el NSGA-II, 534% para el SPEA y para el CNSGA-II 1225%. Claramente el NSGA-II es el más beneficiado al aumentar el número de elementos de búsqueda.

- Considerando la métrica Error, para las corridas donde se utiliza para la búsqueda un único procesador el SPEA es el que obtiene el mejor resultado. Sin embargo cuando se consideran dos o más procesadores es el NSGA-II el que se comporta mejor. Lo mismo ocurre con las métricas distancia generacional y spacing.

- Para la métrica spacing, considerando las corridas a y b, el SPEA obtiene resultados considerablemente mejores que los obtenidos por el CNSGA-II, sin embargo, al aumentar el número de

Relación	CNSGA-II	NSGA-II	SPEA
$(b/a - 1) \cdot 100$	15.2542	8.4112	4.0268
$(c/a - 1) \cdot 100$	296.6102	649.5327	156.3758
$(d/a - 1) \cdot 100$	1225.424	101.6822	534.8993

Tabla 6.5. ZDT1: Ganancia porcentual de N entre los resultados de las corridas paralelas y la secuencial.

elementos de búsqueda, esta diferencia se reduce notablemente y no existe un claro ganador, es más, el CNSGA-II supera ligeramente al SPEA considerando los conjuntos propuestos en las corridas b.

- Los resultados de la métrica ME son muy similares, no existiendo una clara ventaja entre los algoritmos que ocupan los tres primeros lugares. Sin embargo un resultado recurrente es que al considerar un número mayor de procesadores, el SPEA va reduciendo su desempeño con relación al NSGA-II y CNSGA-II para esta métrica.

- Para las métricas ONVG, ONVGR y N, GD y Spacing, claramente se obtienen mejores resultados cuando se utilizan dos o más procesos de búsqueda.

- Considerando las métricas ONVG y ONVGR, los pMOEAs que utilizan dos y cuatro NSGA-II obtienen los mejores resultados, quedando en tercer lugar el SPEA-d. Luego se ubican los CNSGA-II-d y CNSGA-II-c seguido por el SPEA con dos elementos de búsqueda. Luego se ubican el NSGA-II-b seguido por el SPEA-b, NSGA-II-b, NSGA-II-a, CNSGA-IIb y en último lugar el CNSGA-a.

- Para las corridas con un único elemento de búsqueda, teniendo en cuenta la métrica N, el orden es claro, en primer lugar los resultados de los SPEA, en segundo lugar los correspondientes al NSGA-II y en tercer lugar los del CNSGA-II. En cada caso, los resultados de las corridas b superaron a los de las corridas a.

- Con relación a la métrica cobertura, la Tabla 6.9 presenta los resultados obtenidos de la aplicación de dicha métrica sobre los resultados obtenidos por los algoritmos que utilizan elitismo durante el proceso de evolución. Con respecto a esta métrica, en promedio, los resultados obtenidos por las corridas con un número mayor de procesadores es menos cubierto. Además, estos son los que más soluciones cubren, demostrando toda su potencialidad.

- Considerando el tiempo promedio de ejecución, la Tabla 6.8 indica que los MOEAs que obtienen un mejor desempeño en cuanto a la calidad de la aproximación, requieren en general un mayor tiempo de ejecución. De echo, el conjunto de resultados obtenidos por el NSGA-II-d es el mejor de todos, para las distintas métricas consideradas, mientras que el tiempo consumido para obtenerlo es el mayor. Esto se debe fundamentalmente a que el procedimiento de eliminación de elementos cubiertos llevado a cabo en el colector domina el tiempo de ejecución. En problemas de mayor complejidad donde el tiempo de cálculo de la función objetivo sea mayor, el tiempo consumido por dicho procedimiento podría ser relativamente despreciable.

Posición	**Conjunto**	**ONVG**	**Conjunto**	**ONVGR**	**Conjunto**	**N**
1	NSGA-II-d	3356	NSGA-II-d	0.59493	NSGA-II-d	2158
2	NSGA-II-c	2486	NSGA-II-c	0.440702	SPEA-d	946
3	SPEA-d	2436	SPEA-d	0.431838	NSGA-II-c	802
4	CNSGA-II-d	2125	CNSGA-II-d	0.376706	CNSGA-II-d	782
5	CNSGA-II-c	1906	CNSGA-II-c	0.337883	SPEA-c	382
6	SPEA-c	1699	SPEA-c	0.301188	CNSGA-II-c	234
7	NSGA-II-b	1307	NSGA-II-b	0.231697	SPEA-b	155
8	SPEA-b	1247	SPEA-b	0.22106	SPEA-a	149
9	NSGA-II-a	1222	NSGA-II-a	0.216628	NSGA-II-b	116
10	SPEA-a	1195	SPEA-a	0.211842	NSGA-II-a	107
11	CNSGA-II-b	1025	CNSGA-II-b	0.181705	CNSGA-II-b	68
12	CNSGA-II-a	875	CNSGA-II-a	0.155114	CNSGA-II-a	59
13	NSGA-a	100	NSGA-a	0.017727	NSGA-d	3
14	NSGA-d	97	NSGA-d	0.017196	NSGA-c	2
15	NSGA-c	65	NSGA-c	0.011523	FFGA-d	2
16	NSGA-b	43	NSGA-b	0.007623	NSGA-b	1
17	NPGA-d	35	NPGA-d	0.006205	FFGA-c	1
18	FFGA-d	34	FFGA-d	0.006027	FFGA-a	0
19	FFGA-c	21	FFGA-c	0.003723	NPGA-a	0
20	NPGA-c	21	NPGA-c	0.003723	NSGA-a	0
21	FFGA-a	20	FFGA-a	0.003545	FFGA-b	0
22	FFGA-b	16	FFGA-b	0.002836	NPGA-b	0
23	NPGA-a	12	NPGA-a	0.002127	NPGA-c	0
24	NPGA-b	9	NPGA-b	0.001595	NPGA-d	0

Tabla 6.6. ZDT1: Métricas ONVG, ONVGR y N aplicadas a los resultados de las corridas.

Posición	Conjunto	Error	Conjunto	GD	Conjunto	ME	Conjunto	Spacing
1	NSGA-II-d	0.3570	NSGA-II-d	0.0001	NSGA-II-d	0.0009	NSGA-II-d	0.0003
2	SPEA-d	0.6117	SPEA-d	0.0002	SPEA-d	0.0011	NSGA-II-c	0.0004
3	CNSGA-II-d	0.6320	NSGA-II-c	0.0002	CNSGA-II-d	0.0011	SPEA-d	0.0004
4	NSGA-II-c	0.6774	CNSGA-II-d	0.0002	SPEA-c	0.0012	CNSGA-II-d	0.0005
5	SPEA-c	0.7752	SPEA-c	0.0003	NSGA-II-c	0.0014	CNSGA-II-c	0.0005
6	SPEA-a	0.8753	CNSGA-II-c	0.0004	SPEA-a	0.0016	SPEA-c	0.0006
7	SPEA-b	0.8757	SPEA-b	0.0005	NSGA-II-a	0.0016	SPEA-b	0.0008
8	CNSGA-II-c	0.8772	SPEA-a	0.0005	CNSGA-II-a	0.0017	NSGA-II-b	0.0008
9	NSGA-II-b	0.9112	NSGA-II-b	0.0005	CNSGA-II-b	0.0017	SPEA-a	0.0009
10	NSGA-II-a	0.9124	NSGA-II-a	0.0005	SPEA-b	0.0018	CNSGA-II-b	0.0011
11	CNSGA-II-a	0.9326	CNSGA-II-b	0.0005	CNSGA-II-c	0.0018	NSGA-II-a	0.0017
12	CNSGA-II-b	0.9337	CNSGA-II-a	0.0006	NSGA-II-b	0.0019	CNSGA-II-a	0.0025
13	FFGA-d	0.9412	NSGA-d	0.0095	NSGA-d	0.0287	NSGA-a	0.0090
14	FFGA-c	0.9524	NSGA-c	0.0131	NSGA-c	0.0380	NSGA-d	0.0101
15	NSGA-d	0.9691	NSGA-b	0.0164	NSGA-b	0.0535	NSGA-c	0.0180
16	NSGA-c	0.9692	FFGA-d	0.0201	FFGA-d	0.0705	NSGA-b	0.0296
17	NSGA-b	0.9767	NPGA-d	0.0229	NPGA-d	0.0709	FFGA-d	0.0325
18	FFGA-a	1.0000	NSGA-a	0.0249	NPGA-b	0.0793	NPGA-d	0.0374
19	NPGA-a	1.0000	FFGA-c	0.0307	NSGA-a	0.0980	NPGA-a	0.0588
20	NSGA-a	1.0000	NPGA-c	0.0427	FFGA-b	0.1315	NPGA-c	0.0695
21	FFGA-b	1.0000	NPGA-b	0.0480	NPGA-c	0.1557	FFGA-b	0.0720
22	NPGA-b	1.0000	FFGA-b	0.0568	FFGA-c	0.1558	FFGA-c	0.0831
23	NPGA-c	1.0000	NPGA-a	0.2857	NPGA-a	1.5938	NPGA-b	0.1449
24	NPGA-d	1.0000	FFGA-a	0.3959	FFGA-a	5.6699	FFGA-a	0.2470

Tabla 6.7. ZDT1: Métricas Error, GD, ME y Spacing aplicadas a los resultados de las corridas.

Posición	Conjunto	Tiempo (Segundos)
1	NPGA-d	3852.2
2	NSGA-d	4659
3	FFGA-d	4671.8
4	NPGA-c	4698.4
5	NPGA-a	5085.9
6	NSGA-b	5163.7
7	NSGA-a	5271.5
8	FFGA-a	5475
9	NSGA-c	5510.5
10	SPEA-a	5815.7
11	CNSGA-II-a	5855.8
12	NSGA-II-a	6251.5
13	FFGA-b	6744
14	NPGA-b	6802.4
15	FFGA-c	6822.6
16	CNSGA-II-b	7692
17	NSGA-II-b	7760.2
18	CNSGA-II-c	7826.8
19	SPEA-b	7899.2
20	SPEA-c	7920
21	NSGA-II-c	8657.3
22	SPEA-d	17567.5
23	CNSGA-II-d	18965.4
24	NSGA-II-d	31244.7

Tabla 6.8. ZDT1: Tiempo promedio utilizado para obtener cada conjunto aproximación.

Conjunto	CNSGA-II				NSGA-II				SPEA				Promedio
	a	b	c	d	a	b	c	d	a	b	c	d	
CNSGA-II-a	0.000	0.000	0.147	0.047	0.234	0.224	0.052	0.014	0.219	0.210	0.100	0.043	0.108
CNSGA-II-b	0.080	0.000	0.160	0.052	0.252	0.241	0.058	0.016	0.237	0.234	0.110	0.048	0.124
CNSGA-II-c	0.458	0.446	0.000	0.121	0.399	0.400	0.131	0.045	0.364	0.369	0.203	0.112	0.254
CNSGA-II-d	0.697	0.693	0.556	0.000	0.645	0.641	0.333	0.155	0.605	0.608	0.442	0.310	0.474
NSGA-II-a	0.306	0.299	0.194	0.072	0.000	0.000	0.069	0.026	0.235	0.239	0.116	0.061	0.135
NSGA-II-b	0.317	0.318	0.205	0.081	0.046	0.000	0.000	0.027	0.255	0.258	0.124	0.064	0.147
NSGA-II-c	0.663	0.661	0.508	0.267	0.620	0.614	0.000	0.127	0.571	0.577	0.391	0.239	0.437
NSGA-II-d	0.835	0.832	0.725	0.486	0.798	0.796	0.526	0.000	0.753	0.758	0.612	0.474	0.633
SPEA-a	0.305	0.304	0.206	0.075	0.257	0.252	0.076	0.034	0.000	0.000	0.125	0.063	0.141
SPEA-b	0.309	0.315	0.212	0.076	0.260	0.256	0.077	0.034	0.020	0.000	0.128	0.064	0.146
SPEA-c	0.518	0.509	0.369	0.182	0.461	0.461	0.194	0.073	0.446	0.452	0.000	0.158	0.319
SPEA-d	0.662	0.664	0.553	0.290	0.632	0.637	0.323	0.153	0.616	0.621	0.467	0.000	0.468
Promedio	0.429	0.420	0.320	0.146	0.384	0.377	0.159	0.058	0.360	0.361	0.235	0.136	

Tabla 6.9. ZDT1: cobertura entre conjuntos calculados por MOEAs de segunda generación.

	CNSGA-II-a	FFGA-a	NPGA-a	NSGA-II-a	NSGA-a	SPEA-a	Promedio
CNSGA-II-a	0.000000	1.000000	1.000000	0.234861	1.000000	0.219247	0.575685
FFGA-a	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
NPGA-a	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.166667
NSGA-II-a	0.306286	1.000000	1.000000	0.000000	1.000000	0.235146	0.590239
NSGA-a	0.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0.333333
SPEA-a	0.305143	1.000000	1.000000	0.256956	1.000000	0.000000	0.593683
Promedio	0.101905	0.833333	0.666667	0.081969	0.500000	0.075732	

Tabla 6.10. ZDT1: Cobertura entre los conjuntos calculados en las corridas a.

	CNSGA-II-b	FFGA-b	NPGA-b	NSGA-II-b	NSGA-b	SPEA-b	Promedio
CNSGA-II-b	0.000000	0.812500	0.888889	0.241010	0.767442	0.234964	0.490801
FFGA-b	0.000976	0.000000	0.222222	0.000000	0.000000	0.000802	0.037333
NPGA-b	0.000000	0.375000	0.000000	0.000000	0.046512	0.000000	0.070252
NSGA-II-b	0.318049	0.875000	0.888889	0.000000	0.744186	0.258220	0.514057
NSGA-b	0.003902	0.750000	0.777778	0.004591	0.000000	0.003208	0.256580
SPEA-b	0.315122	0.875000	0.888889	0.256312	0.837209	0.000000	0.528755
Promedio	0.106341	0.614583	0.611111	0.083652	0.399225	0.082866	

Tabla 6.11. ZDT1: Cobertura entre los conjuntos calculados en las corridas b.

	CNSGA-II-c	FFGA-c	NPGA-c	NSGA-II-c	NSGA-c	SPEA-c	Promedio
CNSGA-II-c	0.000000	0.809524	0.952381	0.131134	0.892308	0.203649	0.498166
FFGA-c	0.000525	0.000000	0.428571	0.000000	0.046154	0.001177	0.079405
NPGA-c	0.000000	0.238095	0.000000	0.000000	0.107692	0.000000	0.057631
NSGA-II-c	0.508395	0.857143	0.952381	0.000000	0.938462	0.391407	0.607964
NSGA-c	0.001049	0.428571	0.666667	0.000000	0.000000	0.001177	0.182911
SPEA-c	0.369885	0.761905	0.952381	0.194690	0.907692	0.000000	0.531092
Promedio	0.146642	0.515873	0.658730	0.054304	0.482051	0.099568	

Tabla 6.12. ZDT1: Cobertura entre los conjuntos calculados en las corridas c.

	CNSGA-II-d	FFGA-d	NPGA-d	NSGA-II-d	NSGA-d	SPEA-d	Promedio
CNSGA-II-d	0.000000	0.852941	0.914286	0.155244	0.917526	0.310755	0.525125
FFGA-d	0.001412	0.000000	0.257143	0.000000	0.051546	0.001642	0.051957
NPGA-d	0.001412	0.500000	0.000000	0.000298	0.113402	0.000821	0.102655
NSGA-II-d	0.486588	0.941176	0.971429	0.000000	0.969072	0.474959	0.640537
NSGA-d	0.001882	0.705882	0.600000	0.000000	0.000000	0.003695	0.218577
SPEA-d	0.289882	0.882353	0.914286	0.153159	0.907216	0.000000	0.524483
Promedio	0.130196	0.647059	0.609524	0.051450	0.493127	0.131979	

Tabla 6.13. ZDT1: Cobertura entre los conjuntos calculados en las corridas d.

6.4.2. Resultados de los experimentos sobre la función ZDT2

La Tabla 6.15 presenta los valores de las métricas ONVG, ONVGR y N para los conjuntos calculados como aproximaciones del conjunto solución del problema ZDT2 por las distintas corridas de las implementaciones de los algoritmos considerados. La Tabla 6.16 presenta los resultados de las métricas Error, GD, ME y Spacing. La Tabla 6.18 presenta los tiempos de ejecución en segundos de las distintas corridas para el problema ZDT2.

Por su parte, la Tabla 6.17 presenta los resultados de la métrica cobertura sobre los conjuntos de solución proveídos por las corridas de las implementaciones de MOEAs de segunda generación. Finalmente, las tablas 6.19, 6.20, 6.21, 6.22 presentan los resultados de la aplicación de la métrica cobertura entre los conjuntos propuestos por los distintos algoritmos en por las corridas a, b, c y d.

A partir de las distintas tablas se obtienen las siguientes conclusiones parciales:

- La Figura 6.3 presenta los resultados obtenidos para la métrica ONVG por los distintos conjuntos propuestos. Como puede verse, conforme aumenta el número de procesos, para las implementaciones de los algoritmos de segunda generación, aumenta el número de soluciones que proponen. Lo mismo ocurre con los que utilizan NSGA.

- Entre las implementaciones secuenciales, la del NSGA-II es la que obtiene el conjunto con el mayor número de soluciones calculadas.

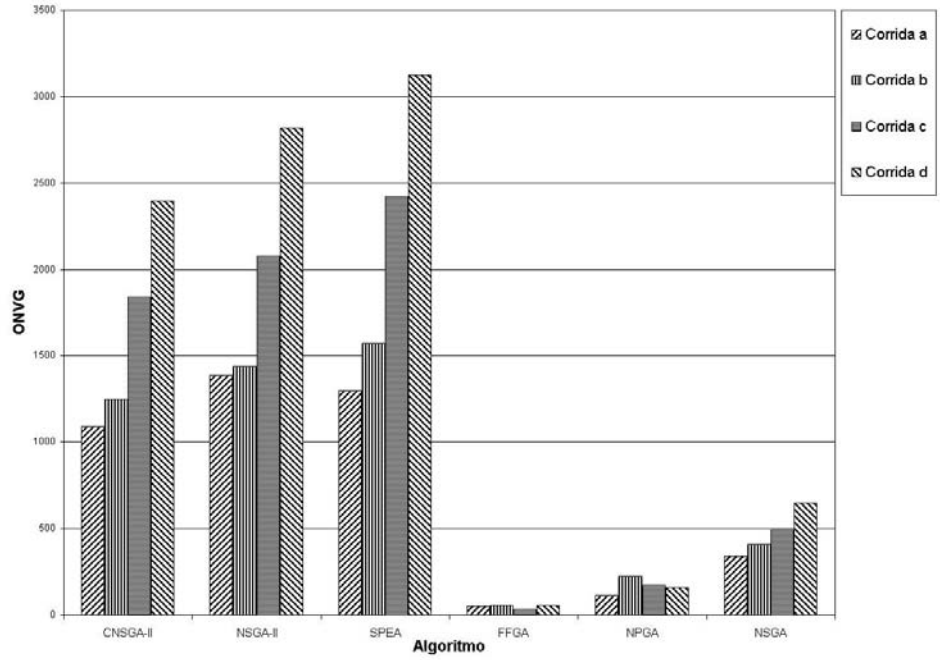

Figura 6.3. ZDT2: Resultado de la métrica ONVG para los distintos conjuntos.

- Al aumentar el número de procesos, son las implementaciones basadas en SPEA las que consiguen mejores resultados para la métrica ONVG. De echo, considerando esta métrica, el mejor de todos los conjuntos es el SPEA-d, obtenido a partir de implementaciones que utilizan cuatro procesos SPEA.

- Entre los conjuntos propuestos por los algoritmos evolutivos multiobjetivo de primera generación utilizados, los que obtienen un mejor valor para la métrica ONVG son los generados utilizando NSGA. Siendo los peores los propuestos por las implementaciones del FFGA. Es interesante apuntar, que para el FFGA el efecto del paralelismo con relación al número de soluciones nodominadas propuestas es prácticamente nulo para el problema y los parámetros utilizados.

- Considerando la métrica ONVGR, el orden de los distintos conjuntos se mantiene igual que con la métrica ONVG.

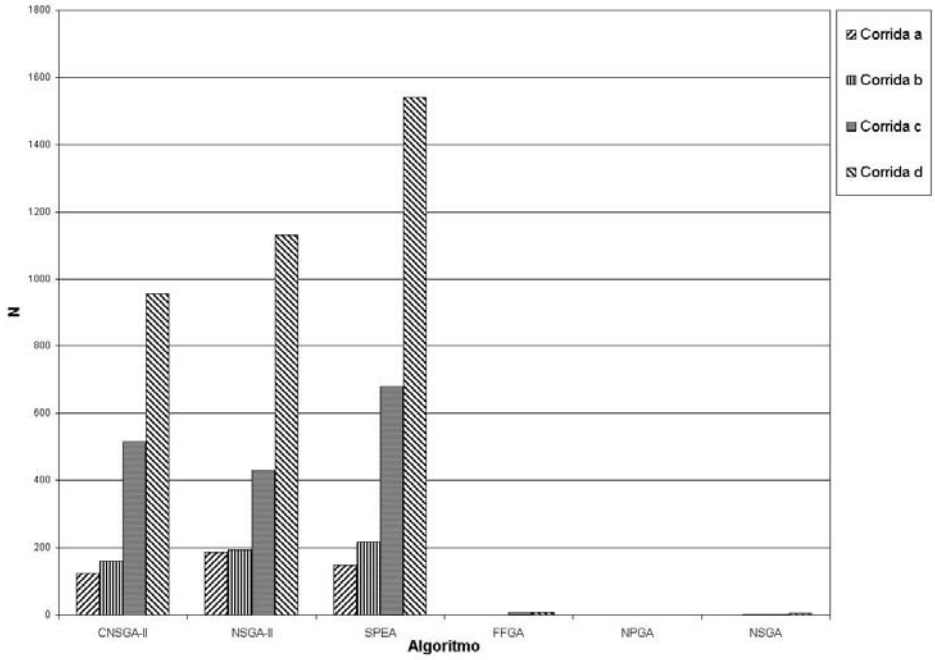

Figura 6.4. ZDT2: Resultado de la métrica N para los distintos conjuntos.

- La Figura 6.4 representa los resultados obtenidos para la métrica N por los distintos conjuntos solución propuestos en las corridas de las implementaciones de los algoritmos considerados.

- Teniendo en cuenta el número de vectores realmente nodominados propuestos por las distintas implementaciones, son los conjuntos obtenidos por las implementaciones que utilizan procesos basados en MOEAs de segunda generación los que obtienen los mejores valores. Los de primera generación, consiguen muy pocas soluciones que verdaderamente pertenecen al conjunto de nodominados. Así los de segunda generación ocupan las posiciones 1 al 12, mientras que los de primera las posiciones 13 al 24.

- Hasta este punto, queda claro que para el problema y parámetros utilizados, las implementaciones que utilizan MOEAs de segunda generación superan a las que pertenecen a la primera generación con respecto tanto al número de soluciones propuestas (ONVG) como en el número de soluciones que verdaderamente son nodominadas.

Relación	CNSGA-II	NSGA-II	SPEA
b/a	1.28	1.037634	1.482993
c/a	4.112	2.306452	4.619048
d/a	7.64	6.069892	10.48299

Tabla 6.14. ZDT2: Razón de N entre los conjuntos obtenidos por las corridas paralelas y la secuencial.

- Los mejores tres conjuntos de soluciones se obtuvieron utilizando implementaciones con cuatro procesos MOEAs. El orden es el siguiente:

  1. SPEA-d,
  2. NSGA-II-d, y
  3. CNSGA-II-d.

- Le siguen a éstas los conjuntos obtenidos por las corridas que utilizan 2 procesos, entre los que nuevamente el conjunto obtenido por la implementación del SPEA es la mejor. A éste, le siguen el propuesto por el CNSGA-II y el NSGA-II. Luego, en la séptima y octava posición aparecen los conjuntos SPEA-b y NSGA-b.

- El conjunto NSGA-II-a es, con respecto a la métrica N, el mejor de todos los conjuntos obtenidos por las corridas de implementaciones secuenciales para este problema y se ubica en la novena posición superando al incluso al CNSGA-II-b.

- La tabla 6.14 presenta la relación entre el número de soluciones realmente nodominadas obtenidas por las corridas paralelas de las implementaciones de los algoritmos de segunda generación. Como puede verse, para el problema considerado al aumentar el número de elementos de proceso, son las implementaciones basadas en SPEA las que obtienen un mayor incremento en el valor de N. Por otro lado, son las basadas en NSGA-II las que consiguen el menor incremento.

- En la Tabla 6.17 puede verse que las soluciones existentes en el conjunto SPEA-d, cubren en promedio el 50 % de las soluciones existentes en los otros conjuntos, siendo además el conjunto

que posee menos soluciones cubiertas en promedio (11%).

- Para la métrica Error, los mejores lugares son ocupados por las implementaciones de los algoritmos de segunda generación con 2 y 4 procesos. En la séptima posición aparece el conjunto solución FFGA-c, sin embargo en este conjunto sólo existen 6 soluciones correctas, mientras que en la octava posición aparece el conjunto SPEA-b con más de 200 soluciones realmente no dominadas. Por lo que, en principio se puede concluir que, el valor de la métrica error es insuficiente para medir la calidad de un conjunto aproximación.

- Con relación a la distancia generacional, en cada corrida, los algoritmos que ocupan los primeros lugares, tienen valores similares.

- Con respecto a la métrica ME, se mantiene que los algoritmos no elitistas logran un desempeño muy pobre con relación al de los algoritmos de segunda generación. Entre estos, es en todos los casos en NSGA el que obtiene un mejor valor para esta métrica para todas las corridas.

- Para la métrica *spacing*, se mantiene que algoritmos elitistas siempre en las primeras posiciones lejos de los algoritmos que no incluyen elitismo. Además, entre los resultados obtenidos por las corridas que utilizan procesos basados en MOEAs elitistas son las que utilizan un número mayor de procesos logran los mejores valores para esta métrica.

- Con respecto al tiempo, son las implementaciones que consiguen el mayor número de soluciones las que requieren mayor tiempo. Es interesante notar que las implementaciones que utilizan NSGA-II y CNSGA-II son más rápidas que las basadas en SPEA.

Posición	Conjunto	ONVG	Conjunto	ONVGR	Conjunto	N
1	SPEA-d	3124	SPEA-d	0.539365	SPEA-d	1541
2	NSGA-II-d	2818	NSGA-II-d	0.486533	NSGA-II-d	1129
3	SPEA-c	2420	SPEA-c	0.417818	CNSGA-II-d	955
4	CNSGA-II-d	2396	CNSGA-II-d	0.413674	SPEA-c	679
5	NSGA-II-c	2079	NSGA-II-c	0.358943	CNSGA-II-c	514
6	CNSGA-II-c	1841	CNSGA-II-c	0.317852	NSGA-II-c	429
7	SPEA-b	1570	SPEA-b	0.271064	SPEA-b	218
8	NSGA-II-b	1436	NSGA-II-b	0.247928	NSGA-II-b	193
9	NSGA-II-a	1386	NSGA-II-a	0.239296	NSGA-II-a	186
10	SPEA-a	1298	SPEA-a	0.224102	CNSGA-II-b	160
11	CNSGA-II-b	1245	CNSGA-II-b	0.214952	SPEA-a	147
12	CNSGA-II-a	1090	CNSGA-II-a	0.188191	CNSGA-II-a	125
13	NSGA-d	647	NSGA-d	0.111706	FFGA-d	6
14	NSGA-c	492	NSGA-c	0.084945	FFGA-c	6
15	NSGA-b	409	NSGA-b	0.070615	NSGA-d	5
16	NSGA-a	337	NSGA-a	0.058184	NSGA-c	2
17	NPGA-b	224	NPGA-b	0.038674	NSGA-d	2
18	NPGA-c	172	NPGA-c	0.029696	FFGA-b	1
19	NPGA-d	158	NPGA-d	0.027279	NPGA-d	1
20	NPGA-a	112	NPGA-a	0.019337	NPGA-c	1
21	FFGA-b	55	FFGA-b	0.009496	NPGA-b	1
22	FFGA-d	55	FFGA-d	0.009496	NSGA-a	1
23	FFGA-a	52	FFGA-a	0.008978	FFGA-a	0
24	FFGA-c	34	FFGA-c	0.005870	NPGA-a	0

Tabla 6.15. ZDT2: Métricas ONVG, ONVGR y N aplicadas a los resultados de las corridas.

Posición	Conjunto	Error	Conjunto	GD	Conjunto	ME	Conjunto	Spacing
1	SPEA-d	0.5067	SPEA-d	0.0002	CNSGA-II-d	0.0015	NSGA-II-d	0.0004
2	NSGA-II-d	0.5994	NSGA-II-d	0.0002	NSGA-II-c	0.0015	SPEA-d	0.0004
3	CNSGA-II-d	0.6014	CNSGA-II-d	0.0002	SPEA-d	0.0015	CNSGA-II-d	0.0005
4	SPEA-c	0.7194	SPEA-c	0.0003	NSGA-II-d	0.0017	SPEA-c	0.0005
5	CNSGA-II-c	0.7208	CNSGA-II-c	0.0003	SPEA-c	0.0018	NSGA-II-c	0.0005
6	NSGA-II-c	0.7937	NSGA-II-c	0.0003	NSGA-II-a	0.0024	CNSGA-II-c	0.0006
7	FFGA-c	0.8235	SPEA-b	0.0004	NSGA-II-b	0.0024	NSGA-II-b	0.0008
8	SPEA-b	0.8611	NSGA-II-b	0.0005	CNSGA-II-c	0.0026	CNSGA-II-b	0.0008
9	NSGA-II-b	0.8656	NSGA-II-a	0.0005	CNSGA-II-b	0.0026	NSGA-II-a	0.0008
10	NSGA-II-a	0.8658	CNSGA-II-b	0.0005	SPEA-a	0.0031	SPEA-b	0.0008
11	CNSGA-II-b	0.8715	SPEA-a	0.0005	SPEA-b	0.0031	CNSGA-II-a	0.0009
12	CNSGA-II-a	0.8853	CNSGA-II-a	0.0006	CNSGA-II-a	0.0031	SPEA-a	0.0009
13	SPEA-a	0.8867	NSGA-d	0.0025	NSGA-d	0.0532	NSGA-d	0.0056
14	FFGA-d	0.8909	NSGA-c	0.0032	NSGA-c	0.0671	NSGA-a	0.0070
15	FFGA-b	0.9818	NSGA-b	0.0039	NSGA-b	0.0864	NSGA-c	0.0078
16	NSGA-d	0.9923	NSGA-a	0.0068	NSGA-a	0.1216	NSGA-b	0.0097
17	NPGA-d	0.9937	NPGA-c	0.0097	NPGA-d	0.2035	NPGA-a	0.0128
18	NPGA-c	0.9942	NPGA-d	0.0101	FFGA-b	0.2489	NPGA-b	0.0146
19	NSGA-b	0.9951	NPGA-b	0.0109	FFGA-c	0.2593	FFGA-a	0.0151
20	NPGA-b	0.9955	FFGA-d	0.0196	FFGA-d	0.2697	NPGA-c	0.0170
21	NSGA-c	0.9959	NPGA-a	0.0207	NPGA-c	0.3198	NPGA-d	0.0187
22	NSGA-a	0.9970	FFGA-c	0.0250	NPGA-b	0.3432	FFGA-d	0.0378
23	FFGA-a	1.0000	FFGA-b	0.0305	NPGA-a	0.3589	FFGA-b	0.0609
24	NPGA-a	1.0000	FFGA-a	0.0562	FFGA-a	0.4585	FFGA-c	0.1199

Tabla 6.16. ZDT2: Métricas Error, GD, ME y Spacing aplicadas a los resultados de las corridas.

Conjunto	CNSGA-II				NSGA-II				SPEA				Promedio
	a	b	c	d	a	b	c	d	a	b	c	d	
CNSGA-II-a	0.000	0.000	0.114	0.062	0.284	0.274	0.166	0.070	0.310	0.268	0.144	0.056	0.146
CNSGA-II-b	0.084	0.000	0.129	0.072	0.323	0.317	0.190	0.082	0.348	0.301	0.161	0.069	0.173
CNSGA-II-c	0.514	0.488	0.000	0.208	0.517	0.508	0.374	0.219	0.558	0.519	0.332	0.175	0.368
CNSGA-II-d	0.652	0.625	0.436	0.000	0.646	0.642	0.529	0.323	0.663	0.617	0.421	0.263	0.485
NSGA-II-a	0.236	0.212	0.095	0.056	0.000	0.000	0.148	0.063	0.280	0.248	0.130	0.054	0.127
NSGA-II-b	0.247	0.223	0.101	0.058	0.022	0.000	0.151	0.065	0.287	0.255	0.136	0.055	0.133
NSGA-II-c	0.438	0.413	0.243	0.143	0.463	0.455	0.000	0.152	0.525	0.480	0.255	0.133	0.308
NSGA-II-d	0.644	0.630	0.414	0.301	0.653	0.651	0.512	0.000	0.697	0.642	0.436	0.256	0.486
SPEA-a	0.230	0.206	0.104	0.059	0.240	0.232	0.133	0.048	0.000	0.000	0.122	0.050	0.119
SPEA-b	0.239	0.220	0.117	0.072	0.253	0.248	0.141	0.053	0.052	0.000	0.139	0.064	0.133
SPEA-c	0.432	0.416	0.242	0.179	0.454	0.445	0.290	0.184	0.495	0.490	0.000	0.158	0.315
SPEA-d	0.628	0.602	0.468	0.344	0.642	0.636	0.502	0.343	0.676	0.664	0.496	0.000	0.500
Promedio	0.362	0.336	0.205	0.130	0.375	0.367	0.261	0.133	0.408	0.374	0.231	0.111	

Tabla 6.17. ZDT2: cobertura entre conjuntos calculados por MOEAs de segunda generación.

Posición	Conjunto	Tiempo (Segundos)
1	NPGA-c	3317.7
2	NPGA-a	3330.8
3	FFGA-d	3766.7
4	NSGA-a	3776.0
5	FFGA-c	3796.4
6	FFGA-a	4143.3
7	CNSGA-II-a	4167.7
8	NSGA-II-a	4192.3
9	NPGA-d	4267.3
10	NSGA-II-b	4990.9
11	CNSGA-II-b	5009.9
12	NSGA-c	5103.9
13	NSGA-II-c	6231.0
14	CNSGA-II-c	6294.5
15	SPEA-a	6298.4
16	FFGA-b	6617.0
17	NPGA-b	6848.7
18	NSGA-b	7405.5
19	SPEA-b	7686.3
20	SPEA-c	10238.0
21	NSGA-d	11867.3
22	CNSGA-II-d	18815.4
23	NSGA-II-d	19874.7
24	SPEA-d	32719.8

Tabla 6.18. ZDT2: Tiempo promedio utilizado para obtener cada conjunto aproximación.

	CNSGA-II-a	FFGA-a	NPGA-a	NSGA-a	NSGA-II-a	SPEA-a	Promedio
CNSGA-II-a	0.000000	1.000000	0.678571	0.635015	0.284271	0.309707	0.484594
FFGA-a	0.000000	0.000000	0.008929	0.000000	0.000000	0.000000	0.001488
NPGA-a	0.000000	0.865385	0.000000	0.000000	0.000000	0.000000	0.144231
NSGA-a	0.000000	1.000000	0.982143	0.000000	0.000000	0.000770	0.330486
NSGA-II-a	0.235780	0.980769	0.598214	0.614243	0.000000	0.280431	0.451573
SPEA-a	0.230275	1.000000	0.767857	0.792285	0.240260	0.000000	0.505113
Promedio	0.077676	0.807692	0.505952	0.340257	0.087422	0.098485	

Tabla 6.19. ZDT2: Cobertura entre los conjuntos calculados en las corridas a.

	CNSGA-II-b	FFGA-b	NPGA-b	NSGA-b	NSGA-II-b	SPEA-b	Promedio
CNSGA-II-b	0.000000	0.963636	0.857143	0.552567	0.316852	0.300637	0.498473
FFGA-b	0.000000	0.000000	0.348214	0.041565	0.000000	0.000000	0.064963
NPGA-b	0.000803	0.836364	0.000000	0.009780	0.000000	0.000000	0.141158
NSGA-b	0.001606	0.927273	0.982143	0.000000	0.000000	0.001911	0.318822
NSGA-II-b	0.223293	0.909091	0.647321	0.540342	0.000000	0.254777	0.429137
SPEA-b	0.220080	0.963636	0.995536	0.955990	0.247911	0.000000	0.563859
Promedio	0.074297	0.766667	0.638393	0.350041	0.094127	0.092887	

Tabla 6.20. ZDT2: Cobertura entre los conjuntos calculados en las corridas b.

	CNSGA-II-c	FFGA-c	NPGA-c	NSGA-c	NSGA-II-c	SPEA-c	Promedio
CNSGA-II-c	0.000000	0.794118	0.988372	0.825203	0.374218	0.331818	0.552288
FFGA-c	0.003259	0.000000	0.523256	0.065041	0.001443	0.003306	0.099384
NPGA-c	0.000543	0.558824	0.000000	0.024390	0.000962	0.000000	0.097453
NSGA-c	0.002716	0.676471	0.965116	0.000000	0.001443	0.000826	0.274429
NSGA-II-c	0.242803	0.764706	0.587209	0.546748	0.000000	0.254545	0.399335
SPEA-c	0.242260	0.794118	0.994186	0.977642	0.290043	0.000000	0.549708
Promedio	0.081930	0.598039	0.676357	0.406504	0.111352	0.098416	

Tabla 6.21. ZDT2: Cobertura entre los conjuntos calculados en las corridas c.

	CNSGA-II-d	FFGA-d	NPGA-d	NSGA-d	NSGA-II-d	SPEA-d	Promedio
CNSGA-II-d	0.000000	0.854545	0.981013	0.795981	0.322924	0.263444	0.536318
FFGA-d	0.003339	0.000000	0.588608	0.058733	0.000710	0.002561	0.108992
NPGA-d	0.001252	0.654545	0.000000	0.027821	0.000000	0.000000	0.113936
NSGA-d	0.002922	0.781818	0.962025	0.000000	0.000710	0.000640	0.291353
NSGA-II-d	0.300501	0.836364	0.721519	0.670788	0.000000	0.256402	0.464262
SPEA-d	0.343907	0.872727	0.993671	0.973725	0.342796	0.000000	0.587804
Promedio	0.108653	0.666667	0.707806	0.421175	0.111190	0.087175	

Tabla 6.22. ZDT2: Cobertura entre los conjuntos calculados en las corridas d.

6.4.3. Resultados de los experimentos sobre la función ZDT3

La Tabla 6.23 presenta los valores de las métricas ONVG, ONVGR y N para los conjuntos solución propuestos, como aproximaciones a la solución del problema ZDT3, por las distintas implementaciones de los algoritmos. La Tabla 6.24 presenta los resultados de las métricas Error, GD, ME y Spacing. La Tabla 6.25 presenta los tiempos promedio de ejecución en segundos de las distintas corridas para el problema ZDT3.

La Tabla 6.26 presenta los resultados de la métrica cobertura comparando los conjuntos de solución proveídos por las implementaciones que utilizan MOEAs de segunda generación. Las tablas 6.27, 6.28, 6.29, 6.30 presentan los resultados de la aplicación de la métrica cobertura entre los conjuntos calculados por los distintos algoritmos en las corridas a, b, c y d.

A partir de los resultados obtenidos, presentados en las distintas tablas, se obtienen las siguientes conclusiones parciales:

- La Figura 6.5 presenta los resultados, obtenidos por los distintos conjuntos propuestos, para la métrica ONVG. Como puede verse, el conjunto de soluciones SPEA-d es el mayor de todos, en segundo lugar se encuentra el NSGA-II-c, seguido por el CNSGA-II-d y el NSGA-II-d.

- El número de soluciones halladas por las implementaciones paralelas basadas en SPEA y CNSGA-II con dos procesos es menor al número de soluciones propuestas por las implementaciones paralelas que utilizan un único proceso MOEA y un colector.

- Considerando los conjuntos encontrados por las corridas a y b son los obtenidos por las implementaciones del CNSGA-II los que obtienen el mayor valor para la métrica ONVG. Sin embargo, para las corridas c el número mayor de soluciones es obtenido por las ejecuciones de las implementaciones del NSGA-II. Finalmente, en las corridas con cuatro procesos es el conjunto SPEA-d el de cardinalidad mayor, siendo además el mayor de todos los conjuntos propuestos.

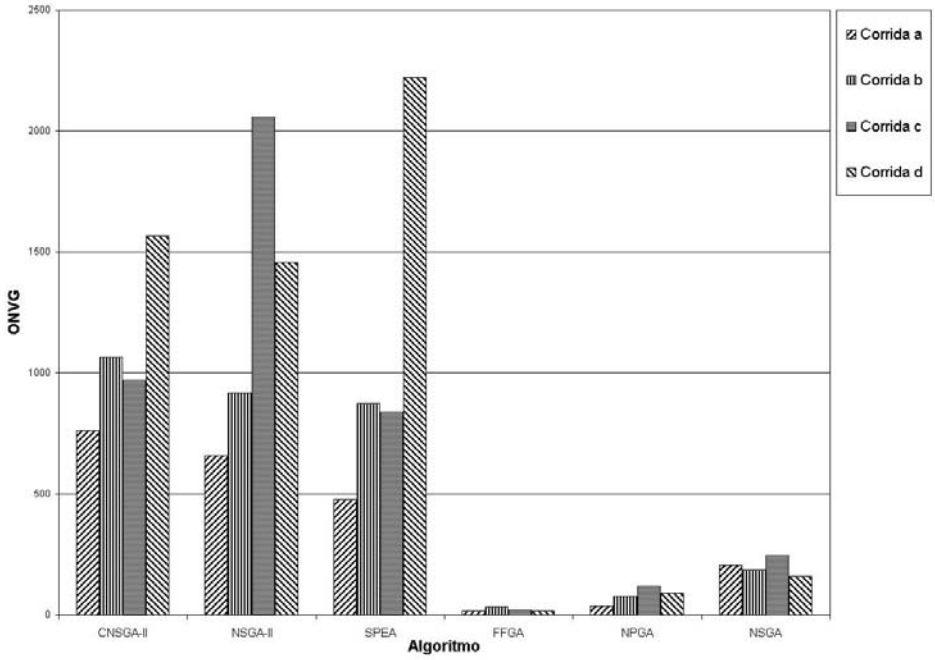

Figura 6.5. ZDT3: Resultado de la métrica ONVG para los distintos conjuntos.

- Para los algoritmos de primera generación, como en las comparaciones anteriores, los conjuntos de soluciones calculados por las distintas implementaciones del NSGA son las que reportan el mayor número de soluciones, seguido por las del NPGA. En cuanto a la cardinalidad del conjunto solución, en el último lugar se encuentran los conjuntos obtenidos por las implementaciones del FFGA.

- Para la métrica ONVGR se mantiene el mismo orden que para la métrica ONVG.

- La Figura 6.6 presenta el resultado de la métrica N para los distintos conjuntos obtenidos. Como se puede visualizar, para esta métrica, el primer y segundo lugar es ocupado por los conjuntos solución obtenidos por las corridas de NSGA-II paralelo con cuatro y dos procesos respectivamente. Son, además, las implementaciones del NSGA-II las que consiguen el mayor número de soluciones verdaderamente nodominadas considerando de forma independiente los conjuntos obtenidos por las corridas a y b.

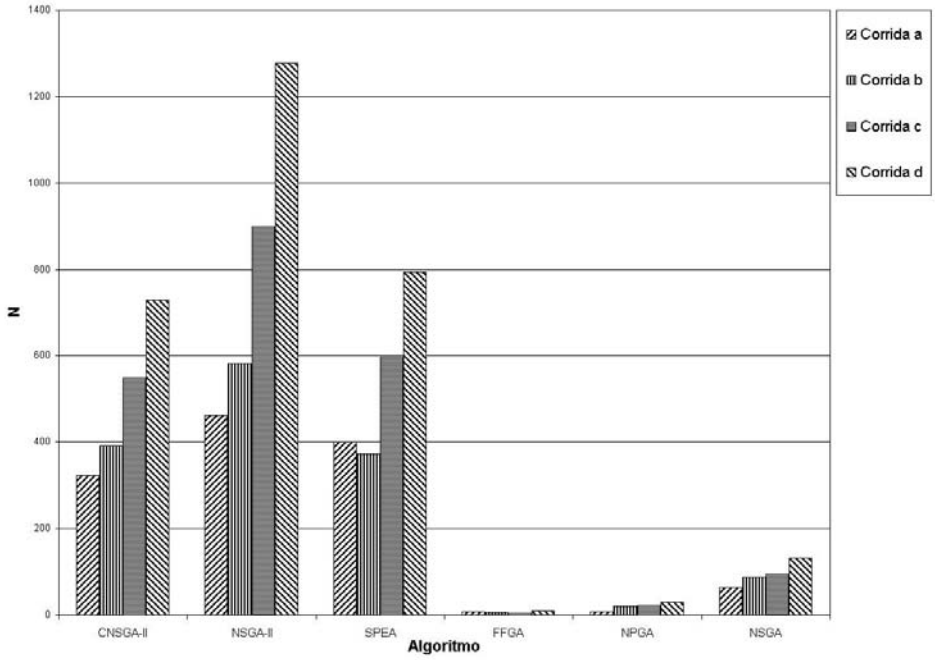

Figura 6.6. ZDT3: Resultado de la métrica N para los distintos conjuntos.

- Puede observarse que, en general, para este problema y el modelo de paralelización utilizado, el número de soluciones propuestas realmente nodominadas aumenta conforme aumenta el número de procesos. No ocurre lo mismo para la métrica ONVG.

- Como en las comparaciones anteriores, con relación al número de soluciones proveídas, las implementaciones de los MOEAs de primera generación son ampliamente superadas por los de segunda generación.

- Para la métrica Error, es el conjunto NSGA-II-d el mejor de todos. Cabe señalar la relatividad de esta métrica ya que según ella el conjunto NSGA-d (posición 3) es mejor que el SPEA-d (posición 17), a pesar que el primero sólo tiene un total de 131 soluciones verdaderamente no-dominadas mientras que el segundo 793. Siendo que además, el 99,1 por ciento de las soluciones del NSGA-d son cubiertas por las soluciones del SPEA-d como se muestra en las tabla 6.30.

Posición	Conjunto	ONVG	Conjunto	ONVGR	Conjunto	N
1	SPEA-d	2222	SPEA-d	0.282733	NSGA-II-d	1278
2	NSGA-II-c	2056	NSGA-II-c	0.261611	NSGA-II-c	900
3	CNSGA-II-d	1565	CNSGA-II-d	0.199135	SPEA-d	793
4	NSGA-II-d	1457	NSGA-II-d	0.185393	CNSGA-II-d	729
5	CNSGA-II-b	1066	CNSGA-II-b	0.135641	SPEA-c	598
6	CNSGA-II-c	969	CNSGA-II-c	0.123298	NSGA-II-b	582
7	NSGA-II-b	917	NSGA-II-b	0.116682	CNSGA-II-c	549
8	SPEA-b	874	SPEA-b	0.11121	NSGA-II-a	461
9	SPEA-c	839	SPEA-c	0.106757	SPEA-a	400
10	CNSGA-II-a	760	CNSGA-II-a	0.096704	CNSGA-II-b	391
11	NSGA-II-a	658	NSGA-II-a	0.083726	SPEA-b	372
12	SPEA-a	476	SPEA-a	0.060568	CNSGA-II-a	323
13	NSGA-c	246	NSGA-c	0.031302	NSGA-d	131
14	NSGA-a	206	NSGA-a	0.026212	NSGA-c	95
15	NSGA-b	187	NSGA-b	0.023794	NSGA-b	88
16	NSGA-d	158	NSGA-d	0.020104	NSGA-a	63
17	NPGA-c	118	NPGA-c	0.015015	NPGA-d	29
18	NPGA-d	88	NPGA-d	0.011197	NPGA-c	23
19	NPGA-b	75	NPGA-b	0.009543	NPGA-b	20
20	NPGA-a	38	NPGA-a	0.004835	FFGA-d	10
21	FFGA-b	32	FFGA-b	0.004072	FFGA-a	8
22	FFGA-c	19	FFGA-c	0.002418	NPGA-a	7
23	FFGA-a	15	FFGA-a	0.001909	FFGA-b	5
24	FFGA-d	15	FFGA-d	0.001909	FFGA-c	4

Tabla 6.23. ZDT3: Métricas ONVG, ONVGR y N aplicadas a los resultados de las corridas.

Posición	Conjunto	Error	Conjunto	GD	Conjunto	ME	Conjunto	Spacing
1	NSGA-II-d	0.1229	NSGA-II-d	0.0016	FFGA-a	0.0670	SPEA-a	0.0171
2	SPEA-a	0.1597	SPEA-c	0.0079	NSGA-d	0.4694	SPEA-d	0.0190
3	NSGA-d	0.1709	SPEA-a	0.0098	SPEA-a	0.4804	NSGA-II-c	0.0204
4	SPEA-c	0.2872	NSGA-d	0.0103	SPEA-c	0.4970	SPEA-c	0.0205
5	NSGA-II-a	0.2994	CNSGA-II-c	0.0154	NSGA-II-d	0.8242	CNSGA-II-b	0.0278
6	FFGA-d	0.3333	CNSGA-II-d	0.0172	CNSGA-II-c	1.1996	CNSGA-II-c	0.0293
7	NSGA-II-b	0.3653	NSGA-II-b	0.0174	NSGA-II-a	1.2971	CNSGA-II-a	0.0315
8	CNSGA-II-c	0.4334	NSGA-II-a	0.0186	FFGA-c	1.3864	CNSGA-II-d	0.0373
9	FFGA-a	0.4667	SPEA-d	0.0196	NPGA-b	1.4037	SPEA-b	0.0373
10	NSGA-b	0.5294	SPEA-b	0.0201	FFGA-d	1.4781	NSGA-II-a	0.0452
11	CNSGA-II-d	0.5342	NSGA-II-c	0.0211	SPEA-b	1.5774	NSGA-II-b	0.0463
12	NSGA-II-c	0.5623	CNSGA-II-b	0.0252	NSGA-II-b	1.5925	FFGA-c	0.0555
13	SPEA-b	0.5744	CNSGA-II-a	0.0275	FFGA-b	1.8549	NSGA-d	0.0595
14	CNSGA-II-a	0.5750	FFGA-a	0.0317	CNSGA-II-d	1.8779	NSGA-a	0.0659
15	NSGA-c	0.6138	NPGA-b	0.0391	CNSGA-II-a	1.9711	NSGA-II-d	0.0665
16	CNSGA-II-b	0.6332	NSGA-c	0.0507	NSGA-b	2.0035	NSGA-c	0.0669
17	SPEA-d	0.6431	NSGA-b	0.0556	NPGA-a	2.0048	NSGA-b	0.0810
18	NPGA-d	0.6705	NSGA-a	0.0604	NSGA-c	2.0056	NPGA-d	0.1064
19	NSGA-a	0.6942	NPGA-c	0.0855	NPGA-d	2.0102	NPGA-a	0.1091
20	NPGA-b	0.7333	FFGA-c	0.0870	NPGA-c	2.0106	NPGA-b	0.1133
21	FFGA-c	0.7895	NPGA-d	0.0947	NSGA-a	2.0133	FFGA-a	0.1159
22	NPGA-c	0.8051	FFGA-d	0.1002	CNSGA-II-b	2.1068	NPGA-c	0.1199
23	NPGA-a	0.8158	NPGA-a	0.1367	NSGA-II-c	2.3554	FFGA-b	0.1761
24	FFGA-b	0.8438	FFGA-b	0.1406	SPEA-d	2.4082	FFGA-d	0.3965

Tabla 6.24. ZDT3: Métricas Error, GD, ME y Spacing aplicadas a los resultados de las corridas.

Posición	Conjunto	Tiempo
1	NPGA-c	3282.5
2	NPGA-a	3460.4
3	NSGA-a	3696
4	FFGA-c	3719.7
5	FFGA-a	3877.7
6	NPGA-d	4184.3
7	FFGA-d	4195.6
8	NSGA-II-a	4208.3
9	CNSGA-II-a	4215.2
10	FFGA-b	4348.5
11	NSGA-b	4469.2
12	NSGA-c	5048.2
13	CNSGA-II-b	5530.3
14	NSGA-II-b	5708.3
15	NPGA-b	6143.3
16	SPEA-a	6848.3
17	SPEA-b	7813.4
18	CNSGA-II-c	10489.2
19	NSGA-d	11167.7
20	SPEA-c	11909.1
21	NSGA-II-c	14116.5
22	SPEA-d	38623.6
23	CNSGA-II-d	40124.6
24	NSGA-II-d	65497.9

Tabla 6.25. ZDT3: Tiempo promedio utilizado para obtener cada conjunto aproximación.

	CNSGA-II				NSGA-II				SPEA				Promedio
Conjunto	**a**	**b**	**c**	**d**	**a**	**b**	**c**	**d**	**a**	**b**	**c**	**d**	
CNSGA-II-a	0.000	0.112	0.022	0.010	0.024	0.012	0.014	0.004	0.042	0.035	0.020	0.010	0.025
CNSGA-II-b	0.000	0.000	0.024	0.018	0.000	0.024	0.020	0.005	0.000	0.047	0.038	0.012	0.016
CNSGA-II-c	0.329	0.371	0.000	0.191	0.112	0.147	0.444	0.014	0.000	0.169	0.063	0.376	0.185
CNSGA-II-d	0.176	0.225	0.000	0.000	0.000	0.000	0.317	0.022	0.000	0.000	0.000	0.260	0.083
NSGA-II-a	0.254	0.348	0.027	0.106	0.000	0.160	0.385	0.008	0.000	0.141	0.037	0.308	0.148
NSGA-II-b	0.212	0.247	0.049	0.063	0.000	0.000	0.337	0.019	0.000	0.031	0.052	0.269	0.107
NSGA-II-c	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.035	0.000	0.000	0.000	0.046	0.007
NSGA-II-d	0.496	0.497	0.222	0.358	0.251	0.238	0.498	0.000	0.000	0.339	0.000	0.497	0.283
SPEA-a	0.499	0.517	0.236	0.369	0.275	0.262	0.507	0.006	0.000	0.432	0.018	0.506	0.302
SPEA-b	0.192	0.233	0.021	0.021	0.000	0.017	0.308	0.008	0.000	0.000	0.035	0.243	0.090
SPEA-c	0.499	0.499	0.223	0.385	0.251	0.238	0.514	0.016	0.002	0.340	0.000	0.512	0.290
SPEA-d	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.035	0.000	0.000	0.000	0.000	0.003
Promedio	0.221	0.254	0.069	0.127	0.076	0.092	0.279	0.014	0.004	0.128	0.022	0.253	

Tabla 6.26. ZDT3: cobertura entre conjuntos calculados por MOEAs de segunda generación.

	CNSGA-II-a	FFGA-a	NPGA-a	NSGA-II-a	NSGA-a	SPEA-a	Promedio
CNSGA-II-a	0.000000	1.000000	0.947368	0.167458	0.773463	0.271127	0.526569
FFGA-a	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
NPGA-a	0.000000	0.660377	0.000000	0.000000	0.000000	0.000000	0.110063
NSGA-II-a	0.305263	1.000000	0.947368	0.000000	0.854369	0.337324	0.574054
NSGA-a	0.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0.333333
SPEA-a	0.241228	1.000000	0.964912	0.151864	0.941748	0.000000	0.549959
Promedio	0.091082	0.776730	0.643275	0.053220	0.428263	0.101408	

Tabla 6.27. ZDT3: Cobertura entre los conjuntos calculados en las corridas a.

	CNSGA-II-b	FFGA-b	NPGA-b	NSGA-II-b	NSGA-b	SPEA-b	Promedio
CNSGA-II-b	0.000000	0.980000	0.995122	0.149232	0.978328	0.328879	0.571927
FFGA-b	0.000000	0.000000	0.302439	0.000000	0.052632	0.000000	0.059178
NPGA-b	0.000000	0.820000	0.000000	0.000000	0.018576	0.000000	0.139763
NSGA-II-b	0.377111	0.980000	0.995122	0.000000	0.981424	0.409646	0.623884
NSGA-b	0.001251	0.860000	0.985366	0.001919	0.000000	0.001162	0.308283
SPEA-b	0.238899	0.980000	0.995122	0.141075	0.984520	0.000000	0.556603
Promedio	0.102877	0.770000	0.712195	0.048704	0.502580	0.123281	

Tabla 6.28. ZDT3: Cobertura entre los conjuntos calculados en las corridas b.

	CNSGA-II-c	FFGA-c	NPGA-c	NSGA-II-c	NSGA-c	SPEA-c	Promedio
CNSGA-II-c	0.000000	0.909091	0.983051	0.178664	0.982927	0.299721	0.558909
FFGA-c	0.003112	0.000000	0.378531	0.002270	0.102439	0.001993	0.081391
NPGA-c	0.000889	0.745455	0.000000	0.000000	0.070732	0.000797	0.136312
NSGA-II-c	0.383281	0.872727	0.994350	0.000000	0.985366	0.390594	0.604386
NSGA-c	0.000889	0.763636	0.949153	0.000649	0.000000	0.000399	0.285788
SPEA-c	0.293908	0.890909	0.983051	0.184501	0.995122	0.000000	0.557915
Promedio	0.113680	0.696970	0.714689	0.061014	0.522764	0.115584	

Tabla 6.29. ZDT3: Cobertura entre los conjuntos calculados en las corridas c.

	CNSGA-II-d	FFGA-d	NPGA-d	NSGA-II-d	NSGA-d	SPEA-d	Promedio
CNSGA-II-d	0.000000	0.860465	0.977273	0.144297	0.987207	0.260726	0.538328
FFGA-d	0.002346	0.000000	0.636364	0.001832	0.061834	0.002100	0.117413
NPGA-d	0.000335	0.418605	0.000000	0.000229	0.070362	0.000600	0.081689
NSGA-II-d	0.459115	0.883721	0.984848	0.000000	0.989339	0.399040	0.619344
NSGA-d	0.001005	0.627907	0.909091	0.000687	0.000000	0.000600	0.256548
SPEA-d	0.381367	0.883721	0.977273	0.183463	0.991471	0.000000	0.569549
Promedio	0.140695	0.612403	0.747475	0.055085	0.516702	0.110511	

Tabla 6.30. ZDT3: Cobertura entre los conjuntos calculados en las corridas d.

6.4.4. Resultados de los experimentos sobre la función ZDT4

La Tabla 6.32 presenta los valores de las métricas ONVG, ONVGR y N para los conjuntos solución propuestos como aproximaciones al conjunto solución del problema ZDT4 por las distintas corridas de las implementaciones de los algoritmos. La Tabla 6.33 presenta los resultados de las métricas Error, GD, ME y Spacing. La Tabla 6.34 presenta los tiempos de ejecución en segundos de las distintas corridas para el problema ZDT4.

Por otro lado, la Tabla 6.35 presenta los resultados de la métrica cobertura comparando los conjuntos de solución proveídos en todas las corridas por las implementaciones de MOEAs de segunda generación utilizados. Finalmente, las tablas 6.36, 6.37, 6.38, 6.39 presentan los resultados de la aplicación de la métrica cobertura entre los conjuntos propuestos por los distintos algoritmos en por las corridas a, b, c y d. A partir de ellas se obtienen las siguientes conclusiones parciales:

- La Figura 6.7 presenta los resultados obtenidos para métrica ONVG por los distintos conjuntos propuestos. Como puede verse, el conjunto de soluciones SPEA-d es el mayor de todos, en segundo lugar se encuentra el CNSGA-II-d, seguido por el SPEA-c y el CNSGA-II-c. El quinto lugar lo ocupa el NSGA-II-d y el sexto lugar el SPEA-b. En el séptimo lugar se tiene al NSGA-d, el primer conjunto solución propuesto por una implementación de MOEAs de primera generación que se ubica para esta métrica entre los valores obtenidos por los algoritmos elitistas. Note además que los conjuntos NSGA-b y NSGA-a superan ambos, para esta métrica, a los conjuntos CNSGA-II-a, CNSGA-II-b, NSGA-II-a y NSGA-II-b.

- Además, el número de soluciones propuestas por las corridas de las implementaciones de los distintos MOEAs en general aumenta conforme aumenta el número de procesos utilizados para la búsqueda de soluciones.

- Para la métrica ONVGR se mantiene el mismo orden en la clasificación que el existente en la

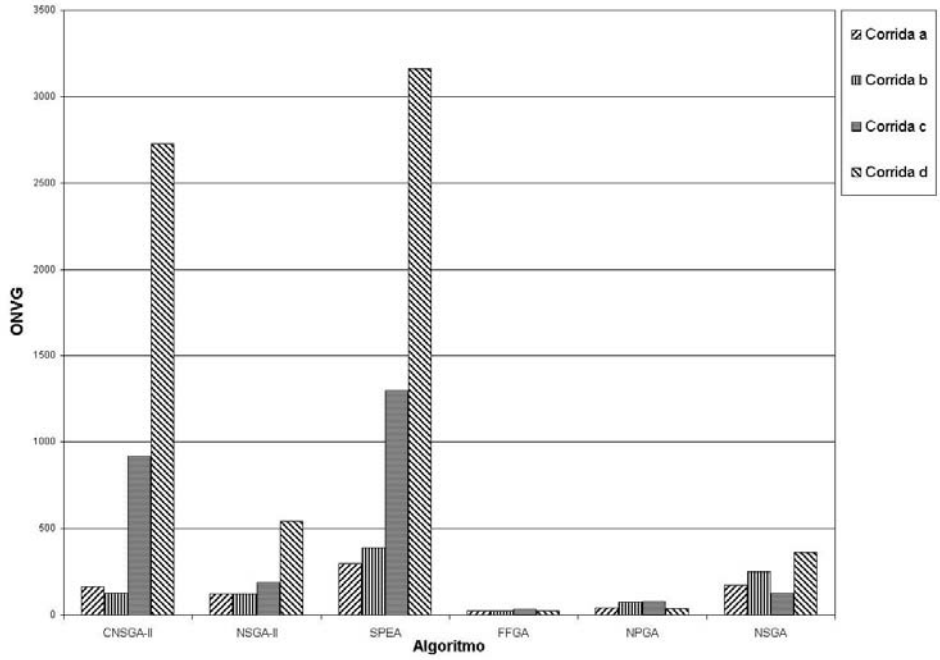

Figura 6.7. ZDT4: Resultado de la métrica ONVG para los distintos conjuntos.

métrica ONVG.

- El número de soluciones existentes en cada conjunto propuesto que son verdaderamente no-dominadas se visualiza en la Figura 6.8. Comparando los resultados para las métricas ONVG y N se puede verificar que, para el problema y los conjuntos analizados, en los primeros 8 lugares existe una relación directa entre la métrica N y ONVG.

- La Tabla 6.31 presenta el valor de N para los conjuntos obtenidos por las implementaciones paralelas de los algoritmos con relación al conjunto correspondiente obtenido por la versión secuencial. Como puede notarse, cuando se utiliza un único procesador, con excepción de las implementaciones del SPEA y el NPGA, se obtiene un mejor valor para N en los conjuntos proveídos por las implementaciones puramente secuenciales. Al aumentar el número de procesos a dos son los procesos que utilizan CNSGA-II los más beneficiados, incrementando en un 1190% el número de soluciones correctas. Igualmente, cuando se utilizan cuatro procesos, el

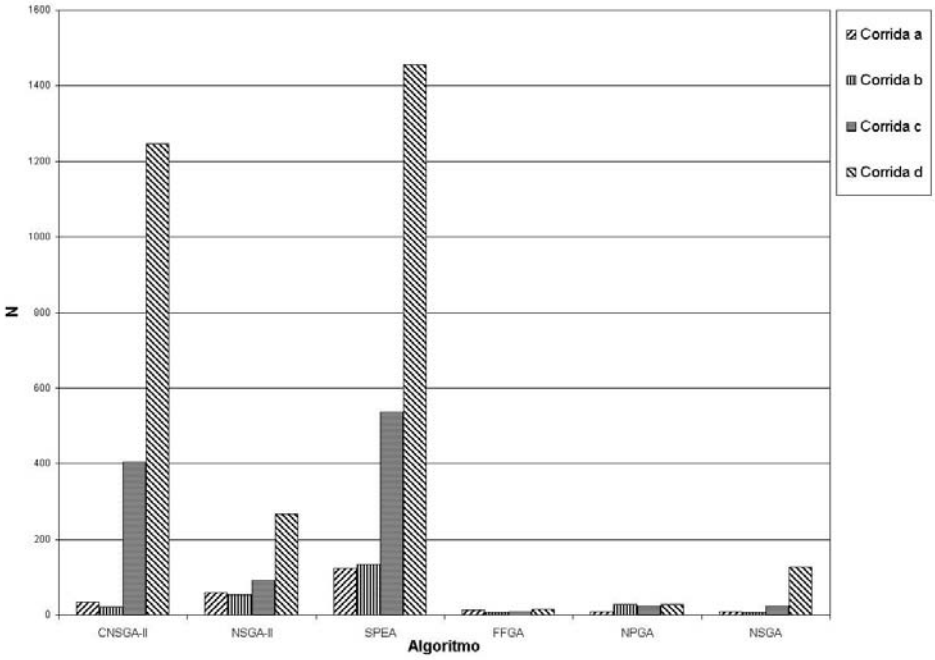

Figura 6.8. ZDT4: Resultado de la métrica N para los distintos conjuntos.

número de soluciones correctas propuestas por el conjunto CNSGA-II-d supera en 36 veces a las del CNSGA-II-a.

- El número de soluciones, verdaderamente nodominadas, propuestas por las corridas paralelas b, c y d, del NPGA son 3, 2,55, y 3 veces más, respectivamente, que las propuestas por implementaciones secuenciales. En la Tabla 6.34, puede verificarse que en promedio entre la corrida a y b del NPGA hay menos de 10 segundos de diferencia.

- Para los resultados obtenidos con los algoritmos de primera generación, como en las comparaciones anteriores, los conjunto de soluciones propuestos por las distintas implementaciones del

	CNSGA-II	**NSGA-II**	**SPEA**	**NSGA**	**NPGA**	**FFGA**
b/a	0.617647	0.883333	1.090164	0.666667	3	0.5
c/a	11.91176	1.533333	4.393443	2.555556	2.555556	0.666667
d/a	36.67647	4.45	11.93443	14.11111	3	1.25

Tabla 6.31. ZDT4: Razón de N entre los conjuntos obtenidos por las corridas paralelas y la secuencial.

NSGA-II son las que reportan el mayor número de soluciones. En segundo lugar se encuentran las soluciones obtenidas por el NPGA. En cuanto a la cardinalidad del conjunto solución, en el último lugar se encuentran los conjuntos obtenidos por las implementaciones del FFGA.

- Con respecto a la métrica error, note que los primeros tres lugares los ocupan conjuntos obtenidos utilizando MOEAs de primera generación. Estos proponen pocas soluciones, pero el porcentaje de éstas que son realmente nodominadas es mayor.

- En este problema, los tiempos promedios de ejecución de las corridas que utilizan MOEAs de primera generación son, en general, menores que los requeridos por MOEAs de segunda generación. Particularmente, las corridas que utilizan NPGA y FFGA requieren un menor tiempo de ejecución que el requerido por cualquier otro MOEA. En el caso de las corridas paralelas que utilizan NPGA, requiriendo un menor tiempo de ejecución que la corrida b del CNSGA-II obtiene un número mayor de soluciones nodominadas.

- Para la métrica Cobertura, los conjuntos proporcionados por todas las corridas que utilizan el CNSGA-II son los que, en promedio, tienen menos soluciones cubiertas. Siendo los conjuntos obtenidos por las implementaciones que utilizan un único proceso CNSGA-II las que cubren en promedio más soluciones.

Posición	Conjunto	ONVG	Conjunto	ONVGR	Conjunto	N
1	SPEA-d	3160	SPEA-d	0.669775	SPEA-d	1456
2	CNSGA-II-d	2726	CNSGA-II-d	0.577787	CNSGA-II-d	1247
3	SPEA-c	1296	SPEA-c	0.274693	SPEA-c	536
4	CNSGA-II-c	920	CNSGA-II-c	0.194998	CNSGA-II-c	405
5	NSGA-II-d	542	NSGA-II-d	0.114879	NSGA-II-d	267
6	SPEA-b	390	SPEA-b	0.082662	SPEA-b	133
7	NSGA-d	362	NSGA-d	0.076727	NSGA-d	127
8	SPEA-a	296	SPEA-a	0.062738	SPEA-a	122
9	NSGA-b	250	NSGA-b	0.052989	NSGA-II-c	92
10	NSGA-II-c	185	NSGA-II-c	0.039212	NSGA-II-a	60
11	NSGA-a	172	NSGA-a	0.036456	NSGA-II-b	53
12	CNSGA-II-a	163	CNSGA-II-a	0.034549	CNSGA-II-a	34
13	CNSGA-II-b	126	CNSGA-II-b	0.026706	NPGA-b	27
14	NSGA-c	126	NSGA-c	0.026706	NPGA-d	27
15	NSGA-II-b	122	NSGA-II-b	0.025858	NPGA-c	23
16	NSGA-II-a	120	NSGA-II-a	0.025435	NSGA-c	23
17	NPGA-c	78	NPGA-c	0.016532	CNSGA-II-b	21
18	NPGA-b	74	NPGA-b	0.015685	FFGA-d	15
19	NPGA-a	44	NPGA-a	0.009326	FFGA-a	12
20	NPGA-d	39	NPGA-d	0.008266	NPGA-a	9
21	FFGA-c	33	FFGA-c	0.006994	NSGA-a	9
22	FFGA-a	24	FFGA-a	0.005087	FFGA-c	8
23	FFGA-b	23	FFGA-b	0.004875	FFGA-b	6
24	FFGA-d	22	FFGA-d	0.004663	NSGA-b	6

Tabla 6.32. ZDT4: Métricas ONVG, ONVGR y N aplicadas a los resultados de las corridas.

Posición	Conjunto	Error	Conjunto	GD	Conjunto	ME	Conjunto	Spacing
1	NPGA-d	0.3077	NPGA-d	0.0160	CNSGA-II-b	0.1751	CNSGA-II-d	0.0135
2	FFGA-d	0.3182	CNSGA-II-b	0.0188	FFGA-b	0.1852	SPEA-d	0.0219
3	FFGA-a	0.5000	NSGA-c	0.0209	NSGA-c	0.2028	SPEA-c	0.0237
4	NSGA-II-a	0.5000	FFGA-b	0.0395	NPGA-d	0.2857	CNSGA-II-c	0.0238
5	NSGA-II-c	0.5027	FFGA-d	0.3463	FFGA-a	35.0189	NSGA-II-d	0.0336
6	NSGA-II-d	0.5074	FFGA-c	0.7999	FFGA-d	58.0086	SPEA-b	0.0368
7	SPEA-d	0.5392	NPGA-a	0.8240	FFGA-c	58.0116	NPGA-b	0.0398
8	CNSGA-II-d	0.5426	CNSGA-II-a	0.8497	NPGA-a	73.0095	SPEA-a	0.0401
9	CNSGA-II-c	0.5598	FFGA-a	0.8541	NPGA-b	110.0063	NSGA-d	0.0458
10	NSGA-II-b	0.5656	NSGA-II-c	0.8571	NPGA-c	116.0060	NSGA-b	0.0494
11	SPEA-c	0.5864	NSGA-a	0.8577	NSGA-II-a	179.0039	CNSGA-II-a	0.0609
12	SPEA-a	0.5878	NPGA-b	0.8621	NSGA-II-b	182.0039	NPGA-c	0.0612
13	NPGA-b	0.6351	NPGA-c	0.8624	NSGA-a	259.0027	CNSGA-II-b	0.0634
14	NSGA-d	0.6492	NSGA-II-a	0.8636	CNSGA-II-a	274.0022	NSGA-c	0.0663
15	SPEA-b	0.6590	NSGA-II-b	0.8637	NSGA-II-c	289.0024	NSGA-a	0.0721
16	NPGA-c	0.7051	SPEA-c	0.8649	NSGA-b	374.0019	NSGA-II-a	0.0787
17	FFGA-b	0.7391	NSGA-b	0.8650	SPEA-a	443.0016	NSGA-II-c	0.0850
18	FFGA-c	0.7576	SPEA-a	0.8651	NSGA-d	542.0013	NSGA-II-b	0.0859
19	CNSGA-II-a	0.7914	NSGA-d	0.8652	SPEA-b	584.0012	FFGA-b	0.1229
20	NPGA-a	0.7955	SPEA-b	0.8653	NSGA-II-d	812.0009	FFGA-a	0.1279
21	NSGA-c	0.8175	NSGA-II-d	0.8655	CNSGA-II-c	1379.0005	NPGA-d	0.1415
22	CNSGA-II-b	0.8333	CNSGA-II-c	0.8657	SPEA-c	1957.0004	NPGA-a	0.2353
23	NSGA-a	0.9477	CNSGA-II-d	0.8659	CNSGA-II-d	4088.0002	FFGA-c	0.2913
24	NSGA-b	0.9760	SPEA-d	0.8958	SPEA-d	5173.0000	FFGA-d	12.3976

Tabla 6.33. ZDT4: Métricas Error, GD, ME y Spacing aplicadas a los resultados de las corridas.

Posición	Conjunto	Tiempo (Segundos)
1	NPGA-a	2143.800049
2	NPGA-c	2152.399902
3	FFGA-a	2339.000000
4	FFGA-c	2408.500000
5	NPGA-d	2412.100098
6	NSGA-a	2494.399902
7	FFGA-d	2559.800049
8	NPGA-b	3144.199951
9	FFGA-b	3465.100098
10	NSGA-b	3665.199951
11	CNSGA-II-a	3869.300049
12	NSGA-II-a	3974.100098
13	NSGA-c	4170.299805
14	NSGA-II-b	4891.299805
15	NSGA-II-c	5061.799805
16	CNSGA-II-b	5150.200195
17	NSGA-II-d	7192.500000
18	SPEA-a	8048.200195
19	SPEA-b	9424.599609
20	NSGA-d	13950.500000
21	SPEA-c	18704.400391
22	CNSGA-II-c	19143.400391
23	SPEA-d	88126.101562
24	CNSGA-II-d	105028.703125

Tabla 6.34. ZDT4: Tiempo promedio utilizado para obtener cada conjunto aproximación.

Conjunto	CNSGA-II				NSGA-II				SPEA				Promedio
	a	b	c	d	a	b	c	d	a	b	c	d	
CNSGA-II-a	0.000	0.159	0.050	0.039	0.333	0.369	0.243	0.166	0.081	0.092	0.074	0.000	0.134
CNSGA-II-b	0.436	0.000	0.500	0.500	0.500	0.500	0.470	0.500	0.503	0.503	0.496	0.490	0.450
CNSGA-II-c	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.028	0.001	0.002
CNSGA-II-d	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.016	0.001
NSGA-II-a	0.245	0.381	0.008	0.001	0.000	0.131	0.011	0.000	0.027	0.013	0.002	0.000	0.068
NSGA-II-b	0.294	0.397	0.008	0.000	0.000	0.000	0.022	0.002	0.024	0.026	0.003	0.000	0.065
NSGA-II-c	0.000	0.556	0.015	0.002	0.067	0.057	0.000	0.033	0.024	0.026	0.008	0.000	0.066
NSGA-II-d	0.000	0.000	0.017	0.002	0.000	0.000	0.000	0.000	0.000	0.000	0.016	0.000	0.003
SPEA-a	0.000	0.000	0.008	0.000	0.000	0.000	0.000	0.015	0.000	0.156	0.012	0.000	0.016
SPEA-b	0.000	0.000	0.007	0.000	0.000	0.000	0.000	0.011	0.000	0.000	0.015	0.000	0.003
SPEA-c	0.000	0.000	0.009	0.006	0.108	0.131	0.054	0.052	0.007	0.018	0.000	0.000	0.032
SPEA-d	0.006	0.040	0.050	0.041	0.333	0.369	0.243	0.166	0.078	0.090	0.079	0.000	0.125
Promedio	0.099	0.071	0.096	0.092	0.147	0.157	0.122	0.117	0.101	0.113	0.100	0.085	

Tabla 6.35. ZDT4: cobertura entre conjuntos calculados por MOEAs de segunda generación.

	CNSGA-II-a	FFGA-a	NPGA-a	NSGA-II-a	NSGA-a	SPEA-a	Promedio
CNSGA-II-a	0.000000	0.291667	0.363636	0.333333	0.267442	0.081081	0.222860
FFGA-a	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
NPGA-a	0.000000	0.000000	0.000000	0.041667	0.023256	0.000000	0.010820
NSGA-II-a	0.245399	0.000000	0.000000	0.000000	0.750000	0.027027	0.170404
NSGA-a	0.006135	0.000000	0.000000	0.000000	0.000000	0.000000	0.001022
SPEA-a	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
Promedio	0.041922	0.048611	0.060606	0.062500	0.173450	0.018018	

Tabla 6.36. ZDT4: Cobertura entre los conjuntos calculados en las corridas a.

	CNSGA-II-b	FFGA-b	NPGA-b	NSGA-II-b	NSGA-b	SPEA-b	Promedio
CNSGA-II-b	0.000000	0.304348	0.554054	0.500000	0.520000	0.502564	0.396828
FFGA-b	0.000000	0.000000	0.500000	0.500000	0.500000	0.500000	0.333333
NPGA-b	0.007937	0.000000	0.000000	0.000000	0.000000	0.002564	0.001750
NSGA-II-b	0.396825	0.000000	0.000000	0.000000	0.704000	0.025641	0.187744
NSGA-b	0.015873	0.000000	0.000000	0.000000	0.000000	0.002564	0.003073
SPEA-b	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
Promedio	0.070106	0.050725	0.175676	0.166667	0.287333	0.172222	

Tabla 6.37. ZDT4: Cobertura entre los conjuntos calculados en las corridas b.

	CNSGA-II-c	FFGA-c	NPGA-c	NSGA-II-c	NSGA-c	SPEA-c	Promedio
CNSGA-II-c	0.000000	0.000000	0.000000	0.000000	0.000000	0.027778	0.004630
FFGA-c	0.023913	0.000000	0.320513	0.145946	0.015873	0.019290	0.087589
NPGA-c	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
NSGA-II-c	0.015217	0.000000	0.115385	0.000000	0.571429	0.007716	0.118291
NSGA-c	0.500000	0.363636	0.564103	0.470270	0.000000	0.496142	0.399025
SPEA-c	0.008696	0.000000	0.179487	0.054054	0.000000	0.000000	0.040373
Promedio	0.091304	0.060606	0.196581	0.111712	0.097884	0.091821	

Tabla 6.38. ZDT4: Cobertura entre los conjuntos calculados en las corridas c.

	CNSGA-II-d	FFGA-d	NPGA-d	NSGA-II-d	NSGA-d	SPEA-d	Promedio
CNSGA-II-d	0.000000	0.000000	0.000000	0.000000	0.000000	0.016139	0.002690
FFGA-d	0.093544	0.000000	0.025641	0.228782	0.267956	0.080696	0.116103
NPGA-d	0.500367	0.045455	0.000000	0.500000	0.502762	0.490190	0.339796
NSGA-II-d	0.002201	0.000000	0.000000	0.000000	0.000000	0.000316	0.000420
NSGA-d	0.000734	0.000000	0.000000	0.011070	0.000000	0.000000	0.001967
SPEA-d	0.041086	0.000000	0.025641	0.166052	0.226519	0.000000	0.076550
Promedio	0.106322	0.007576	0.008547	0.150984	0.166206	0.097890	

Tabla 6.39. ZDT4: Cobertura entre los conjuntos calculados en las corridas d.

6.5. Conclusiones experimentales

En este capítulo se ha presentado un conjunto de métricas utilizadas para medir el desempeño de MOEAs [36, 86, 103]. Así mismo, se han definido formalmente las funciones de prueba ZDT1 a ZDT4 [99]. Para cada una de estas funciones, se han implementado versiones secuenciales y paralelas de los MOEAs presentados en el Capítulo 4. Los resultados obtenidos por cada implementación, para cada problema considerado, han sido comparados en base a las métricas de desempeño presentadas. A partir de los diferentes resultados expuestos y analizados se puede concluir que:

- No existe una única medida para medir el desempeño de MOEAs, siendo la comparación de MOEAs un problema multiobjetivo. Sin embargo es posible concluir, a partir de los resultados presentados, que el número de soluciones verdaderamente nodominadas (N) obtenidas por un MOEA es, de las métricas consideradas, una de las más significativas.

- Para determinar cual es, entre las implementaciones realizadas, la mejor de todas, la Tabla 6.40 presenta, para cada problema, un ordenamiento lexicográfico de las mismas en base a dos objetivos:

  1. la clasificación por nodominancia (*ranking*) considerando las métricas ONVG, ONVGR, N, Error, GD, ME y Spacing de forma simultánea,
  2. el valor de la métrica N.

  Estos objetivos se han considerado pues son suficientes para establecer un orden completo sobre los algoritmos de segunda generación.

- Para simplificar el análisis de los resultados, en la Tabla 6.41 se presenta la frecuencia en cada posición, del orden lexicográfico antes descrito, de cada MOEA e implementación considerando los cuatro problemas. Así se puede visualizar que en el primer lugar se encuentran dos

implementaciones paralelas del SPEA y dos implementaciones paralelas del NSGA-II y que todas ellas corresponden a las corridas "d" donde se utilizan cuatro procesos. Además, en la segunda posición se encuentran dos implementaciones del NSGA-II, una del CNSGA-II y una del SPEA; de éstas tres, corresponden a implementaciones que utilizan cuatro procesos y una que utiliza dos. En consecuencia, se observa que el paralelismo tiene un doble efecto positivo: por un lado, permitiría reducir los tiempos de ejecución usando más procesadores, y por el otro mejora la calidad de las soluciones obtenidas.

- De los resultados presentados en las Tablas 6.40 y 6.41 queda claro que existe una clara diferencia entre el desempeño obtenido por las implementaciones de los MOEAs que utilizan elitismo y los que no lo utilizan. Además, puede notarse el efecto positivo del paralelismo ya que las implementaciones que utilizan un mayor número de procesos son las que, en general, se encuentran en las primeras posiciones, por lo que en general, se recomienda utilizar implementaciones paralelas que implementan elitismo.

- Entre las implementaciones de primera generación son las del NSGA las que se encuentran mejor posicionadas, pero en general, son superados por algoritmos de la segunda generación.

ZDT1			ZDT2			ZDT3			ZDT4		
Conjunto	**Ranking**	**N**	**Conjunto**	**Ranking**	**N**	**Conjunto**	**Ranking**	**N**	**Conjunto**	**Ranking**	**N**
NSGA-II-d	1	1	SPEA-d	1	1	NSGA-II-d	1	1	SPEA-d	1	1
SPEA-d	2	2	NSGA-II-d	1	2	NSGA-II-c	1	2	CNSGA-II-d	1	2
NSGA-II-c	2	3	CNSGA-II-d	1	3	SPEA-d	1	3	SPEA-c	1	3
CNSGA-II-d	3	4	SPEA-c	2	4	CNSGA-II-d	1	4	CNSGA-II-c	1	4
SPEA-c	4	5	NSGA-II-c	2	6	SPEA-c	1	5	NSGA-II-d	1	5
CNSGA-II-c	4	6	CNSGA-II-c	3	5	NSGA-II-b	1	6	SPEA-b	1	6
SPEA-b	5	7	NSGA-II-b	3	8	CNSGA-II-c	1	7	NSGA-d	1	7
SPEA-a	5	8	NSGA-II-a	3	9	SPEA-a	1	9	SPEA-a	1	8
NSGA-II-b	5	9	SPEA-b	4	7	CNSGA-II-b	1	10	NSGA-II-c	1	9
NSGA-II-a	5	10	CNSGA-II-b	4	10	NSGA-d	1	13	NSGA-II-a	1	10
CNSGA-II-b	6	11	SPEA-a	4	11	FFGA-a	1	21	NSGA-II-b	1	11
CNSGA-II-a	6	12	FFGA-c	4	13	NSGA-II-a	2	8	CNSGA-II-a	1	12
NSGA-d	7	13	CNSGA-II-a	5	12	SPEA-b	2	11	NPGA-d	1	13
FFGA-d	7	14	FFGA-d	6	13	CNSGA-II-a	2	12	NPGA-b	1	13
NSGA-a	7	16	NSGA-c	6	14	NSGA-c	3	14	NPGA-c	1	14
NSGA-c	8	14	NSGA-d	6	15	NSGA-b	3	15	NSGA-c	1	14
FFGA-c	8	15	FFGA-b	6	16	NSGA-a	3	16	CNSGA-II-b	1	15
NSGA-b	9	15	NSGA-b	7	15	NPGA-b	3	19	FFGA-a	1	17
NPGA-d	10	16	NSGA-a	7	16	FFGA-d	3	20	NPGA-a	1	18
NPGA-c	11	16	NPGA-d	7	16	FFGA-c	3	24	NSGA-a	1	18
FFGA-b	11	16	NPGA-c	7	16	NPGA-d	4	17	FFGA-c	1	19
NPGA-b	11	16	NPGA-b	8	16	NPGA-c	4	18	FFGA-b	1	20
NPGA-a	11	16	NPGA-a	8	17	NPGA-a	4	22	NSGA-b	1	20
FFGA-a	12	16	FFGA-a	9	17	FFGA-b	4	23	FFGA-d	2	16

Tabla 6.40. Orden lexicográfico de los MOEAs para cada problema.

Posición en el orden	**Algoritmo**						**Corrida**			
	NSGA-II	CNSGA-II	SPEA	NSGA	FFGA	NPGA	a	b	c	d
1	2	0	2	0	0	0	0	0	0	4
2	2	1	1	0	0	0	0	0	1	3
3	1	1	2	0	0	0	0	0	2	2
4	0	3	1	0	0	0	0	0	2	2
5	2	0	2	0	0	0	0	0	3	1
6	1	2	1	0	0	0	0	2	2	0
7	1	1	1	1	0	0	0	2	1	1
8	1	0	3	0	0	0	4	0	0	0
9	2	1	1	0	0	0	0	3	1	0
10	2	1	0	1	0	0	2	1	0	1
11	1	1	1	0	1	0	2	2	0	0
12	1	2	0	0	1	0	3	0	1	0
13	0	1	1	1	0	1	1	1	0	2
14	0	1	0	0	2	1	1	1	0	2
15	0	0	0	3	0	1	1	0	3	0
16	0	0	0	4	0	0	0	1	2	1
17	0	1	0	1	2	0	1	2	1	0
18	0	0	0	2	1	1	1	3	0	0
19	0	0	0	1	1	2	2	0	0	2
20	0	0	0	1	1	2	1	0	2	1
21	0	0	0	0	2	2	0	1	2	1
22	0	0	0	0	1	3	0	3	1	0
23	0	0	0	1	0	3	3	1	0	0
24	0	0	0	0	4	0	2	1	0	1

Tabla 6.41. Frecuencia de los MOEAs en cada posición para cada problema.

Capítulo 7

Comparación de pMOEAs aplicados al Problema de la Programación Óptima de Estaciones de Bombeo de Agua

Los sistemas de distribución de agua son piezas claves de la infraestructura urbana e indispensables para el desarrollo económico sustentable. El crecimiento constante de la demanda hace que estos sistemas se vuelvan grandes y complejos por lo que a fin de minimizar el costo operativo de los mismos es necesario analizarlo en base a sus componentes principales. Las estaciones de bombeo de agua son componentes fundamentales de estos sistemas. La programación óptima de las bombas que las componen ha probado ser un método práctico y efectivo para reducir los costos operativos sin modificar la infraestructura existente, asegurando la provisión de agua incluso en situaciones de demanda alta. De acuerdo al número de variables y objetivos considerados, el problema de optimización de la programación de bombas se vuelve complejo, especialmente en sistemas grandes. Por la relevancia del problema, varios autores lo han estudiado proponiendo distintos enfoques para su resolución [59, 61, 65, 75, 78, 80, 92].

Típicamente, una estación de bombeo está compuesta por un conjunto de bombas de diferente capacidad, las cuales son utilizadas para bombear agua a uno o más reservorios. Éstas trabajan de forma combinada para bombear la cantidad de agua necesaria. Al hacerlo se deben cumplir de forma adecuada con ciertas restricciones hidráulicas y técnicas, tales como el nivel máximo en el reservorio y la potencia de las bombas. En un instante en particular, algunas bombas están trabajando mientras otras no. En este contexto, *programar o planificar la operación de las bombas* significa: elegir de forma adecuada la combinación correcta de bombas que se utilizarán en cada intervalo de tiempo del horizonte de planificación [61]. Entonces, *una programación óptima de bombas* establece la combinación de bombas ha utilizar en cada intervalo de tiempo del horizonte de planificación que, satisfaciendo las restricciones del problema, optimiza los objetivos establecidos [61].

Al principio, se utilizó programación lineal, no lineal, dinámica, mixta, entre otras para la optimización de un único objetivo: el costo de la energía eléctrica. Una revisión completa de estos trabajos puede encontrarse en [65]. Las no-linealidades del sistema físico, las restricciones de los recursos, las restricciones operacionales y el tamaño del espacio de búsqueda hacen que, incluso considerando los objetivos de forma independiente, el problema de la programación óptima de bombas (PPOB) sea computacionalmente intratable utilizando métodos tradicionales de investigación de operaciones. Por lo tanto, se han desarrollado algunas técnicas de aproximación para encararlo. Así, en [61] se presenta un algoritmo genético para la optimización del costo de la energía eléctrica. Además, en [75] se propone la hibridización de los GAs con un método de búsqueda local para la optimización de dos objetivos (costo de energía eléctrica y costo de mantenimiento de bombas). Por otra parte, en [78] se presenta un algoritmo genético simple utilizando GAs considerando las restricciones del sistema estableciendo penalidades. Todos los métodos evolutivos anteriormente citados utilizan modificaciones del algoritmo genético simple presentado en [50].

Los algoritmos evolutivos han probado ser herramientas útiles para la resolución de problemas

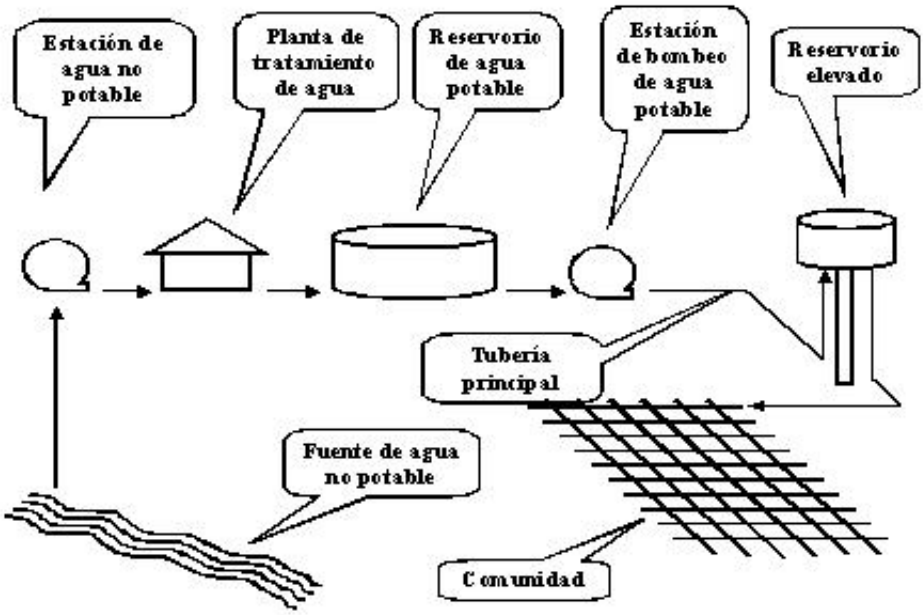

Figura 7.1. Sistema de distribución de agua básico.

de programación de bombas considerando un único objetivo [61, 75, 78]. Consecuentemente, en [80] se propone la utilización de MOEAs para la resolución de un problema de programación óptima de bombas considerando cuatro objetivos de forma simultanea y la utilización de un algoritmo heurístico para manejar las restricciones del problema planteado.

Los objetivos considerados en [80] son: costo de la energía eléctrica, costo de mantenimiento, pico máximo de potencia y nivel de variación en el reservorio. Posteriormente, en [92] se presentan y comparan distintos MOEAs para la optimización del problema considerado en [80]. Si bien los resultados obtenidos con los MOEAs secuenciales son aceptables, para aplicaciones reales, es necesario un incremento en la eficiencia y la eficacia. Para obtener este incremento, en [91] se propone el uso y desarrollo de pMOEAs y se presentan y comparan los resultados obtenidos al utilizar diferentes pMOEAs para enfrentar el problema multiobjetivo de programación óptima de estaciones de bombeo planteado en [80]. En el presente trabajo se propone la resolución del problema considerado utilizando un modelo de paralelización distinto al presentado en [91]. El objetivo es contrastar la calidad del con-

junto aproximación calculado utilizando implementaciones paralelas de MOEAs conforme al modelo de paralelización presentado en la Sección 5.2, con respecto a las versiones secuenciales.

7.1. Definición del problema

7.1.1. Modelo hidráulico

En [92] se presenta un análisis exhaustivo del problema de la programación de bombeo. En la presente sección, se señalan algunas de las consideraciones más importantes a partir de las cuales se desarrolló la formulación matemática del problema y son:

- Considerar un modelo hidráulico simplificado, basado en la estación de bombeo principal de Asunción y con la demanda de agua como único dato externo al modelo.
- El modelo utilizado es similar al utilizado en [75] y está compuesto de:
  - una fuente de agua inagotable: el reservorio de agua potable,
  - una estación de agua potable compuesta por n_b bombas utilizadas para bombear agua desde la fuente de agua hasta un reservorio elevado,
  - una tubería principal es utilizada para llevar agua desde la estación de bombeo hasta el reservorio elevado, y
  - un reservorio elevado, el cual provee agua a la comunidad.

  El modelo hidráulico utilizando 5 bombas se presenta en la Figura 7.2.
- Se puede suponer que la fuente proporciona toda el agua que sea necesaria en cualquier momento y sin costos adicionales; así como que las condiciones de presión máxima y mínima en las tuberías son satisfechas cualquiera sea el nivel en el tanque.

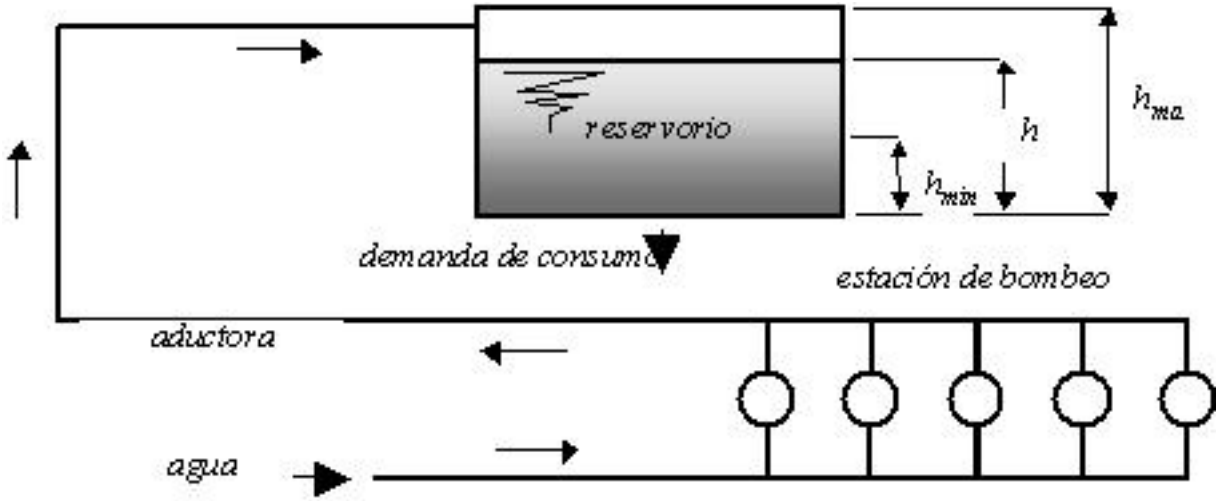

Figura 7.2. Modelo utilizado con 5 bombas.

- La estación de bombeo posee n_b bombas centrifugas de distintas velocidades, cada una de ellas con velocidad constante, trabajando de forma asociada. Las capacidades de las bombas se suponen constantes en cada intervalo de tiempo. Por tanto, para cada intervalo de una hora, cada combinación de bombas tiene valores de descarga, consumo de energía eléctrica y potencia fijos. De esta manera, las no-linealidades en la combinación de las bombas se consideran utilizando una tabla con las características de cada combinación de bombas (ver Tabla 7.1).

- Por ser la práctica usual, es conveniente considerar un período de optimización de un día, dividido en 24 intervalos de 1 hora.

- Se debe definir un modelo matemático del problema que contemple simultáneamente: el costo de la energía eléctrica consumida, el costo de mantenimiento de las bombas, la diferencia de nivel en el reservorio entre el inicio y final del período de optimización y la potencia máxima alcanzada.

- El costo de la energía eléctrica consumida debe considerar la estructura tarifaría ya que los costos de consumo de energía eléctrica se reducen si la programación de bombeo óptima establece la

Hora	1					...	24				
Bomba	B1	B2	B3	B4	B5	...	B1	B2	B3	B4	B5
Bit	0	1	1	0	1	...	1	0	0	0	1
Significado	apagada	encendida	encendida	apagada	encendida	...	encendida	apagada	apagada	apagada	encendida

Figura 7.3. Representación de una programación óptima.

menor cantidad posible de bombas encendidas en el horario de tarifa elevada.

- Para estimar el costo de mantenimiento de las bombas es frecuente utilizar el número de encendidos [59]. La estimación debe considerar posibles encendidos entre el día anterior y el actual, de forma a facilitar el uso periódico de la programación.

- Los niveles de agua máximo y mínimo en el reservorio deben ser considerados como restricciones del modelo.

7.1.2. Representación

Para representar la programación de las bombas, se utiliza una codificación binaria. En cada intervalo de tiempo, se utiliza un bit para indicar si una bomba se encuentra trabajando o no. Un cero representa una bomba que no se encuentra trabajando, mientras que un uno representa una bomba que se encuentra encendida. Se considera un periodo de optimización de un día dividido en 24 intervalos de una hora. Entonces, para este modelo, las bombas sólo pueden ser encendidas o apagadas al comienzo de cada intervalo de tiempo. La Figura 7.3 presenta el modelo de representación considerando 5 bombas y 24 intervalos de una hora.

7.1.3. Formulación matemática de los objetivos del problema

A fin de definir el problema de programación óptima de bombas multiobjetivo, en las siguientes secciones se presentan los distintos objetivos de optimización, luego se presenta una definición formal del problema en el contexto de optimización multiobjetivo.

7.1.3.1. Costo de la energía eléctrica (E_c)

El costo de la energía eléctrica significa el costo total debido al consumo de energía eléctrica por parte de todas las bombas que componen la estación de bombeo y que son utilizadas durante el período de optimización considerado [61, 75].

Cuando se analiza el costo por consumo de energía eléctrica, la estructura tarifaria de la compañía eléctrica es una cuestión importante. Generalmente, el costo de la energía eléctrica no es el mismo durante todo el día. De echo, existen horas de mayor demanda de electricidad, como horas de menor demanda. Por tanto, el costo de la energía eléctrica varía. Este trabajo considera la siguiente estructura tarifaria:

- Una tarifa reducida (C_l): la cual va desde las 0 : 00 hasta las 17 : 00 horas, y luego desde las 22 : 00 hasta las 24 : 00 horas.
- Una tarifa elevada (C_h): desde las 17 : 00 hasta las 22 : 00 horas.

La influencia de esta estructura de tarifación variable en el calendarizado de las bombas es sustancial. El costo de la energía eléctrica puede ser drásticamente reducido si la programación óptima de las bombas establece el número menor de bombas, o las bombas con un menor consumo, trabajando durante el período de tarifa alta. De esta forma, el agua almacenada en los reservorios puede ser utilizada durante este período de tiempo para proveer agua a la comunidad. La ecuación 7.1 proporciona la

expresión matemática del costo de la energía eléctrica E_c.

$$E_c = C_l \sum_{t=1}^{17} c(p_t) + C_h \sum_{t=18}^{22} c(p_t) + C_l \sum_{t=23}^{24} c(p_t) \tag{7.1}$$

donde

t : intervalo de tiempo considerado

p_t : combinación de bombas en el intervalo t ($p_t \in \{0,1\}^{n_b}$)

n_b : número de bombas en la estación

$c(p_t)$: energía eléctrica utilizada por la combinación de bombas p_t en el intervalo de tiempo t.

7.1.3.2. Costo del mantenimiento de las bombas (N_s)

El costo de mantenimiento de las bombas puede ser tan importante, e incluso mayor, que el costo de la energía eléctrica. En Lansey et al. [59] se propone el concepto del número de encendidos como una forma alternativa de medir el costo de mantenimiento de las bombas. En otras palabras, el tiempo de vida útil de una bomba puede ser medido de forma indirecta a partir del número de veces que ésta ha sido encendida.

Se considera que una bomba se enciende en un intervalo dado t si ha estado apagada en el intervalo $t-1$. Esto es, no se considera como encendido el que una bomba que ya ha estado trabajando en el intervalo anterior continúe encendida en el intervalo siguiente. El número total de encendidos N_s es calculado de forma simple contabilizando el número de encendidos de las bombas en cada intervalo de tiempo como se muestra en la Ecuación 7.2. En esta ecuación, el número de encendidos entre el último intervalo de tiempo del periodo de optimización precedente (el día anterior) y el primer intervalo del día analizado, también es computado. Sin embargo, solo la mitad de tal cantidad es

agregada, suponiendo que existe cierta periodicidad entre programaciones consecutivas.

$$\mathsf{N_s} = \sum_{\mathsf{t}=1}^{24} |max\{0; (p_{\mathsf{t}} - p_{\mathsf{t}-1})\}| + \frac{|max\{0; p_1 - p_{24}\}|}{2} \tag{7.2}$$

7.1.3.3. Variación del nivel en el reservorio (Δh)

Existen diferentes niveles a considerar en el reservorio:

- Un nivel mínimo: el cual garantiza la presión suficiente en la tubería. Este nivel también debe mantenerse por razones de seguridad, puesto que eventos inesperados, como un incendio, pueden consumir grandes cantidades de agua en un intervalo de tiempo pequeño.
- Un nivel máximo: el cual no puede ser excedido a fin de evitar pérdidas en la tubería. Este nivel también debe ser compatible con las dimensiones del reservorio.
- Un nivel inicial: el cual debe ser recuperado al final del periodo de optimización a fin de mantener una programación periódica si las condiciones en días consecutivos no cambian de forma sustancial.

La variación de nivel entre el comienzo y el final del período de optimización Δh, se propone como un objetivo, el cual debe ser minimizado como se muestra en la ecuación 7.3. Sin embargo, los niveles máximos y mínimos deben ser considerados como restricciones. Así, al final de cada intervalo de tiempo, el nivel de agua debe estar entre el nivel máximo h_{max} y el nivel mínimo h_{min} como se plantea en la Ecuación 7.4.

$$\Delta\mathsf{h} = \frac{\sum_{\mathsf{t}=1}^{24}[D(p_{\mathsf{t}}) - d_{\mathsf{t}}]}{S} \tag{7.3}$$

$$\mathsf{h_t} = \frac{\mathsf{h}_{i-1} + [D(p_{\mathsf{t}}) - d_{\mathsf{t}}]}{S} \tag{7.4}$$

sujeto a

$$h_t \leq h_{max}$$

$$h_t \geq h_{min}$$

donde

S : superficie del reservorio, asumida constante

$D(p_t)$: agua bombeada en el intervalo t utilizando la combinación de bombas p_t

d_t : demanda de agua en el intervalo de tiempo t.

7.1.3.4. Pico máximo de potencia (P_{max})

Algunas compañías de electricidad cobran a sus clientes principales de acuerdo a una potencia reservada. Esta potencia reservada tiene un precio fijo, pero un costo adicional elevado o penalidad, se agrega si se excede la potencia contratada. La penalidad se computa utilizando el pico máximo de potencia alcanzado durante el tiempo considerado para la facturación. Por tanto, reducir tal penalidad es muy importante. Este trabajo encara éste problema considerando el pico máximo de potencia diario P_{max} como otro objetivo a minimizar. Este es computado fácilmente utilizando la ecuación 7.5.

$$P_{max} = max[P(p_t)] \tag{7.5}$$

donde:

$P(p_t)$: potencia en el intervalo de tiempo t utilizando la combinación de bombas p_t.

7.1.4. Problema de optimización de bombas multiobjetivo

Con cada uno de los cuatro objetivos considerados definidos formalmente, el problema de optimización multiobjetivo de la programación de bombeo puede definirse como:

$$\text{Minimizar } \mathbf{y} = \mathbf{F}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), f_3(\mathbf{x}), f_4(\mathbf{x})) \tag{7.6}$$

sujeto a

$\mathsf{h}_\mathsf{t} = h(x_\mathsf{t}) \leq \mathsf{h}_max$

$\mathsf{h}_\mathsf{t} = h(x_\mathsf{t}) \geq \mathsf{h}_min$, para cada intervalo t

h_t: nivel en el reservorio al final del intervalo t (ver Ecuación 7.4)

donde:

f_1 : costo de la energía eléctrica utilizada (ver Ecuación 7.1)

f_2 : número de encendidos de las bombas (ver Ecuación 7.2)

f_3 : costo de la energía eléctrica consumida (ver Ecuación 7.3)

f_4 : pico máximo de potencia (ver Ecuación 7.5)

$\mathbf{x} \in \mathcal{X} \subseteq B^{24 \cdot b_n}$ es el vector de decisión, $B \in \{0,1\}$

$\mathbf{y} = (y_1, y_2, y_3, y_4) \in \mathcal{Y} \subset \mathbb{R}^4$ es el vector objetivo

7.1.5. Datos necesarios para la resolución del problema

Las características del modelo utilizado en éste trabajo están basados en la estación principal de bombeo de agua potable de Asunción, como se describe:

- El sistema cuenta con un conjunto de $b_n = 5$ bombas cuyas características técnicas se presentan en la Tabla 7.1.

- Se considera un reservorio elevado con las siguientes dimensiones:

Combinación de bombas	Código	Caudal [mş/h]	Potencia [kW]	Combinación de bombas	Código	Caudal [mş/h]	Potencia [kW]
0	0	0	0	16	10000	1800	595
1	1	1800	595	17	10001	3600	1190
2	10	828	260	18	10010	2620	855
3	11	2600	855	19	10011	4420	1450
4	100	828	260	20	10100	2620	855
5	101	2600	855	21	10101	4420	1450
6	110	1650	520	22	10110	3450	1115
7	111	3450	1115	23	10111	5250	1710
8	1000	1440	445	24	11000	3235	1040
9	1001	3235	1040	25	11001	5035	1635
10	1010	2260	705	26	11010	4060	1300
11	1011	4060	1300	27	11011	5860	1895
12	1100	2260	705	28	11100	4060	1300
13	1101	4060	1300	29	11101	5860	1895
14	1110	3090	965	30	11110	4890	1560
15	1111	4890	1560	31	11111	6690	2155

Tabla 7.1. Características técnicas de las combinaciones de bombas utilizadas.

- Área 2600 $metros^2$,
- $h_{max} = 7$ *metros*,
- $h_{min} = 1$ *metros*,
- $h_{inicial} = 3$ *metros*,
- Volumen 15600 $metros^3$.

- La curva de demanda utilizada se muestra en la Figura 7.4.
- Una estructura tarifaria con $C_H = 2C_L$ es asumida.

7.2. Descripción de los experimentos

Para la resolución del problema de optimización de bombas se han utilizado implementaciones secuenciales y paralelas de los MOEAs presentados en el Capítulo 4. Las implementaciones paralelas se han realizado siguiendo el modelo propuesto en la Sección 5. El entorno paralelo utilizado fue presentado en la Tabla 6.1.

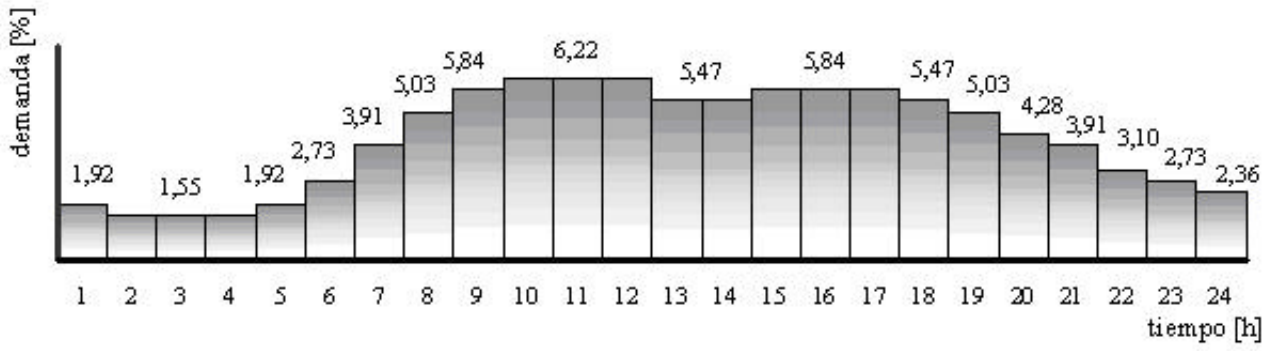

Figura 7.4. Curva de demanda utilizada.

Como en experimentos anteriores del Capítulo 6, se llevaron a cabo 10 ejecuciones distintas para cada una de las implementaciones secuenciales de los algoritmos utilizados. Los parámetros presentados en la tabla 6.2 fueron también utilizados en este problema. Además, se llevaron a cabo 10 ejecuciones paralelas utilizando 1, 2, 4, 8 y 16 procesos MOEAs. Los resultados de las distintas ejecuciones, con número de procesos distinto, se agruparon obteniendo un conjunto de nodominados para cada corrida. Se utilizó la misma notación de comparaciones del Capítulo 6 para identificar los distintos conjuntos propuestos y las corridas con 8 y 16 procesos se nombraron como corridas e y f respectivamente. En las distintas corridas paralelas, se utilizaron para controlar la migración los mismos parámetros presentados en la Tabla 6.3.

7.3. Resultados experimentales y análisis

Los resultados de las métricas utilizadas, sobre los conjuntos propuestos para la resolución del problema de la programación óptima de bombas por las distintas corridas de los algoritmos implementados, se presentan en formato tabular al final de la presente sección. La Tabla 7.3 presenta los valores de las métricas ONVG, ONVGR y N. En la Tabla 7.2 se presentan los resultados de las métricas Error, GD,

ME y Spacing. En la Tabla 7.4 se presentan los tiempos promedio de ejecución, medidos en segundos, de las distintas corridas.

Además, las tablas 7.5, 7.6, 7.7, 7.8, 7.9 y 7.10 presentan información sobre la cobertura de los conjuntos propuestos considerando cada corrida de forma separada. Finalizando, las tablas 7.11 y 7.12 presentan un resumen estadístico de las métricas ONVG y N sobre los conjuntos solución propuestos por cada ejecución de las implementaciones de MOEAs de segunda generación, agrupados por cada corrida. Además, la Tabla 7.13 presenta un resumen estadístico de los tiempos.

A partir de los resultados presentados en las distintas tablas, se obtienen las siguientes conclusiones parciales:

- La Figura 7.5 ilustra los distintos valores de la métrica ONVG para cada conjunto solución considerado. Estos valores se corresponden con los presentados en la tabla 7.3. Como puede verse, los conjuntos propuestos por implementaciones paralelas del NSGA obtienen altos valores para esta métrica. Es más, de los cinco primeros lugares tres están ocupados por conjuntos obtenidos con implementaciones de este algoritmo, compitiendo directamente contra el SPEA. Note sin embargo, que el conjunto NSGA-a ocupa una lejana posición 32. Quedando claro el efecto del paralelismo para la métrica considerada y este algoritmo. En general, al aumentar el número de procesadores el valor de esta métrica ha aumentado.

- Para la métrica ONVGR, los primeros lugares son para los conjuntos NSGA-e y NSGA-f. Si bien el valor mayor lo obtiene el conjunto NSGA-f, el primer lugar para esta métrica está ocupado por el NSGA-e puesto que es el que tiene un valor más cercano a uno, esto es, el conjunto NSGA-e tiene un número de soluciones más cercano al existente en PF_{true}. El echo que los conjuntos NSGA-e y NSGA-f obtengan un valor mayor a uno indica que proponen más soluciones que las existentes en el conjunto PF_{true}. Como PF_{true} se forma a partir de las soluciones nodominadas considerando todos los conjuntos obtenidos, un valor mayor a 1 para NSGA-f y NSGA-e indica

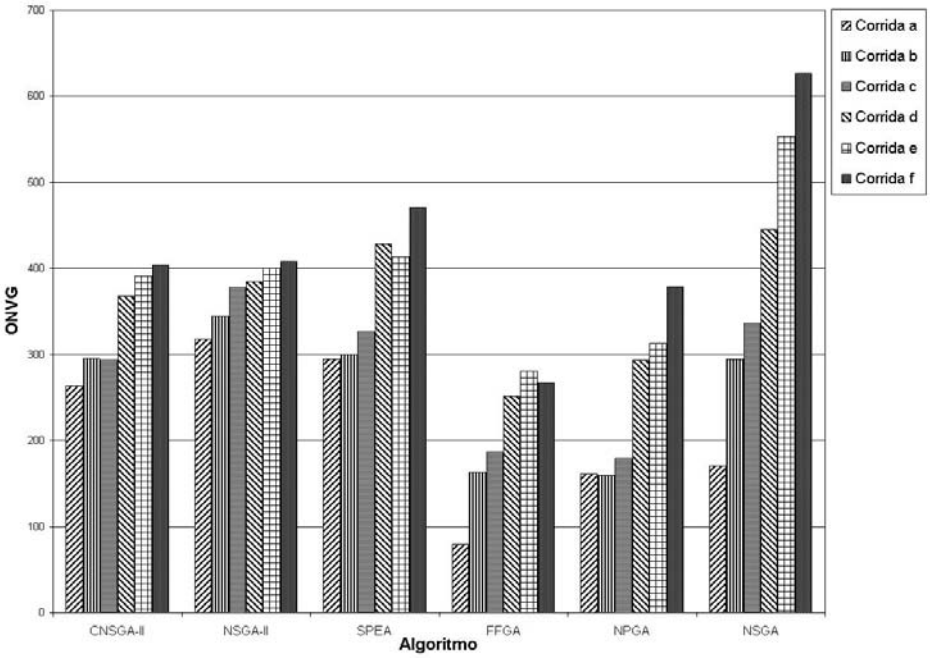

Figura 7.5. Resultado de la métrica ONVG para los distintos conjuntos propuestos.

que existen soluciones en estos conjuntos que no pertenecen a PF_{true}.

- La Figura 7.6 presenta los resultados obtenidos para la métrica N para los conjuntos calculados por cada corrida. Como puede notarse, si bien las implementaciones de los algoritmos de primera generación fueron capaces de proponer un gran número de soluciones, ninguna de ellas forma parte del frente de no dominados.

- El conjunto SPEA-f es el que más soluciones verdaderamente nodominadas posee. Sin embargo, la corrida secuencial del SPEA es la que menos soluciones propone. Los resultados obtenidos tanto por el CNSGA-II como el NSGA-II se demuestran muy competitivos con relación a la métrica N. Incluso, los obtenidos por las implementaciones con un único proceso de búsqueda.

- En cuanto al Error, son los conjuntos NSGA-II-e y NSGA-II-d los mejores. En esta métrica, los algoritmos de primera generación, al no proponer ninguna solución verdaderamente nodominada, obtienen todos un valor de uno.

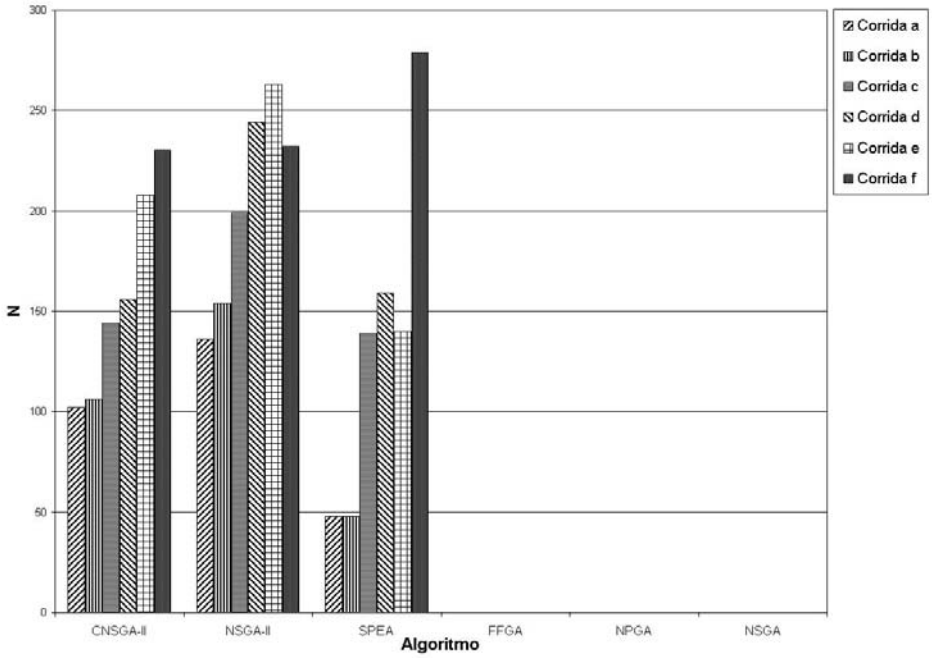

Figura 7.6. Resultado de la métrica N para los distintos conjuntos propuestos.

- Con respecto a las métricas GD, ME y Spacing, los resultados obtenidos por las distintas métricas indican claramente la superioridad de los algoritmos que utilizan elitismo para la obtención de un gran número de soluciones verdaderamente no dominadas.

- Para la métrica Spacing, se puede notar la primacía de las implementaciones basadas en NSGA-II y CNSGA-II con relación a las implementaciones del SPEA. Esta situación, aunque en menor medida, también se cumple para la métrica ME.

- Con respecto al tiempo, la tabla 7.4 presenta los tiempos promedio, medidos en segundos. Como puede visualizarse, los conjuntos solución calculados por implementaciones del SPEA son los que más tiempo requieren para su culminación. Entre estos también se encuentra el conjunto NSGA-f, el cual según la Tabla 7.3 halla un elevado número de soluciones.

A partir de las conclusiones anteriores, queda claro que los conjuntos solución propuestos por los MOEAs de segunda generación son considerablemente mejores que los de primera generación, en

particular teniendo en cuenta la generación de vectores realmente nodominados. Por ello, en este punto se ha considerado oportuno explorar la manera en que cada conjunto solución calculado por cada corrida de las implementaciones del NSGA-II, CNSGA-II y SPEA es obtenido. A tal efecto, se considera la aplicación de las métricas ONVG y N sobre los conjuntos obtenidos por cada ejecución que compone una corrida.

En las tablas 7.11, 7.12, ubicadas al final de la sección, se presentan los valores promedio, la mediana, el mínimo, el máximo y la desviación estándar para las métricas ONVG y N sobre los conjuntos propuestos por las 10 ejecuciones que se llevaron a cabo con cada MOEA de segunda generación. Del mismo modo, en la Tabla 7.13 se presentan estos valores considerando los tiempos de ejecución. Así, se obtienen las siguientes conclusiones:

- La Figura 7.7 ilustra los valores promedio de N considerando las distintas ejecuciones que conforman cada corrida. A partir del mismo notamos que las ejecuciones paralelas, en promedio, obtienen un número mayor de soluciones verdaderamente nodominadas con relación a las secuenciales. Es más, en general, este número aumenta conforme aumenta el número de elementos de proceso.

- Además, se consigue visualizar que en promedio las ejecuciones del NSGA-II y CNSGA-II respectivamente son mejores que las del SPEA. Sin embargo, de acuerdo con la Tabla 7.3 y la Figura 7.6, considerando la corrida completa, la corrida de la implementación paralela del SPEA con 16 procesos es la que más soluciones realmente nodominadas reporta. Esto indica que, utilizando 16 elementos de proceso, existe una cierta ventaja a favor del SPEA con relación a la variedad de soluciones proveídas por ejecuciones diferentes con relación a al NSGA-II y CNSGA-II.

- En la Tabla 7.12, se puede notar que para los algoritmos analizados, los valores máximos y

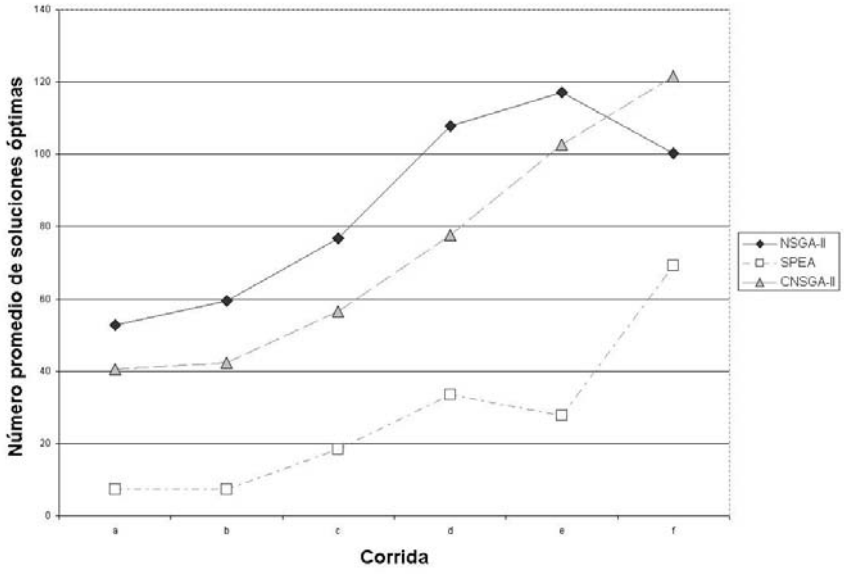

Figura 7.7. Promedio de soluciones realmente nodominadas encontradas en cada corrida.

mínimos del número de soluciones nodominadas calculadas por las implementaciones paralelas son mayores o iguales que los valores correspondientes a las implementaciones secuenciales. En general, estos valores crecen conforme aumenta el número de elementos de proceso.

- En todas las ejecuciones con más de un proceso NSGA-II o CNSGA-II se han obtenido elementos verdaderamente nodominados. Esto no ocurre para las implementaciones paralelas del SPEA.

- En la Tabla 7.4, se pudo visualizar que entre los conjuntos solución propuestos por los algoritmos de segunda generación, son los obtenidos utilizando SPEA los que más tiempo han requerido. Analizando los valores mínimos y máximos para los tiempos de cada ejecución, se puede determinar además que el mínimo tiempo requerido por una ejecución cualquiera del SPEA es superior al máximo tiempo empleado por cualquier otra implementación tanto del NSGA-II como del CNSGA-II.

- En la Figura 7.8, se visualiza como el tiempo promedio de ejecución aumenta conforme aumenta

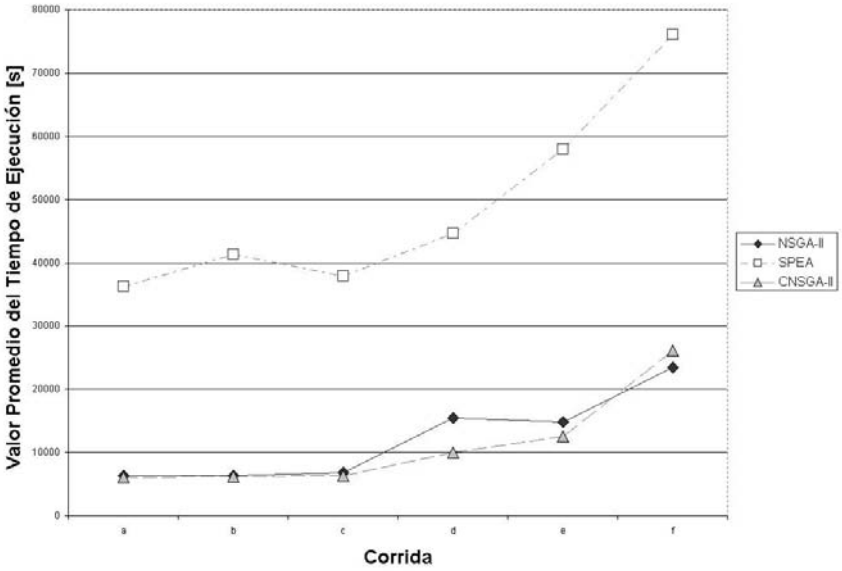

Figura 7.8. Tiempo promedio de ejecución con relación a las corridas realizadas.

el número de procesos utilizados en la búsqueda. Note que tanto para el NSGA-II como para el CNSGA-II, los tiempos de ejecución apenas sufren un incremento. Mientras que en el caso del SPEA, el incremento es mayor.

- En las Figuras 7.9 y 7.10, se presentan los tiempos promedios de ejecución con respecto al número de soluciones propuestas (ONVG) y al número promedio de éstas que verdaderamente son nodominadas. Puede notarse que estos tiempos están aumentan conforme aumenta el número de soluciones manejadas. Note que, en general, las implementaciones paralelas en promedio son capaces de obtener un número mayor de soluciones verdaderamente no dominadas mientras se mantiene un tiempo de ejecución comparable al de corridas secuenciales.

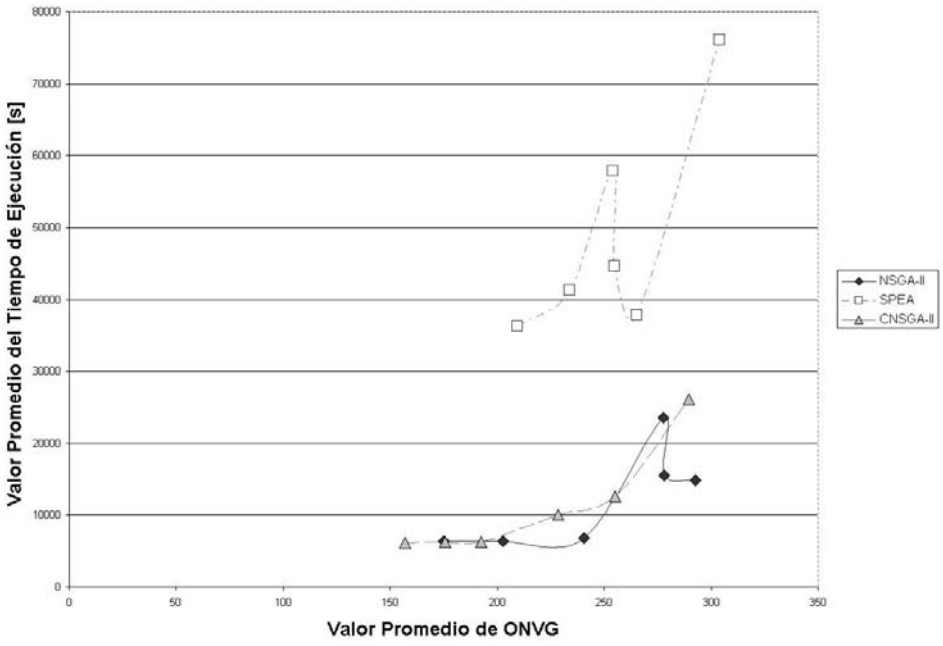

Figura 7.9. Tiempo promedio de ejecución con respecto al valor promedio de ONVG.

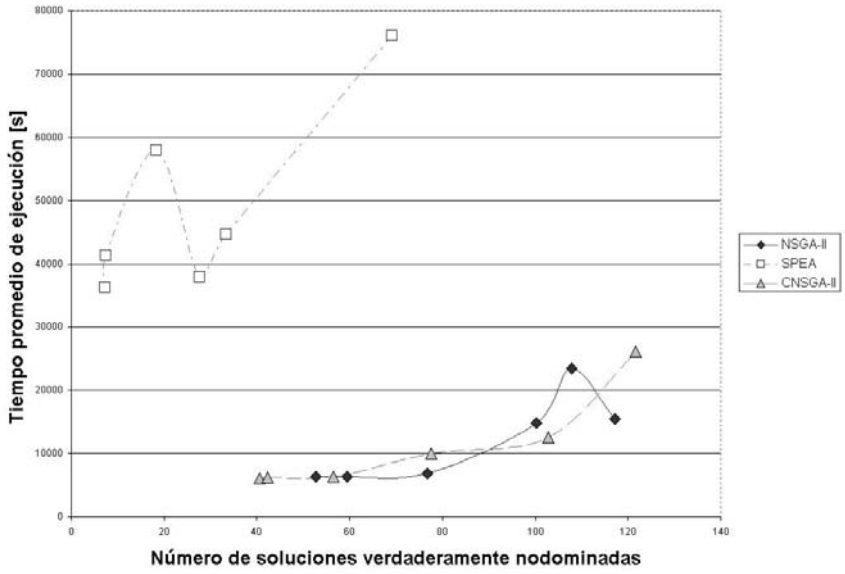

Figura 7.10. Tiempo promedio de ejecución con respecto al número promedio de soluciones nodominadas encontradas.

Posición	Conjunto	Error	Conjunto	GD	Conjunto	ME	Conjunto	Spacing
1	NSGA-II-e	0.3425	NSGA-II-e	0.0036	NSGA-II-c	0.0872	NSGA-II-c	0.0174
2	NSGA-II-d	0.3646	NSGA-II-f	0.0039	CNSGA-II-f	0.1036	CNSGA-II-f	0.0179
3	SPEA-f	0.4076	NSGA-II-d	0.0040	CNSGA-II-c	0.1077	NSGA-II-d	0.0195
4	CNSGA-II-f	0.4307	CNSGA-II-f	0.0042	NSGA-II-f	0.1133	NSGA-II-f	0.0198
5	NSGA-II-f	0.4314	SPEA-f	0.0043	NSGA-II-a	0.1220	CNSGA-II-d	0.0205
6	CNSGA-II-e	0.4680	NSGA-II-c	0.0045	NSGA-II-b	0.1292	NSGA-II-a	0.0224
7	NSGA-II-c	0.4735	CNSGA-II-e	0.0048	SPEA-b	0.1337	NSGA-II-e	0.0231
8	CNSGA-II-c	0.5085	CNSGA-II-c	0.0051	SPEA-a	0.1407	NSGA-II-b	0.0234
9	NSGA-II-b	0.5523	CNSGA-II-d	0.0053	SPEA-f	0.1408	CNSGA-II-c	0.0237
10	NSGA-II-a	0.5710	NSGA-II-b	0.0054	SPEA-e	0.1445	CNSGA-II-e	0.0249
11	SPEA-c	0.5749	NSGA-II-a	0.0055	NSGA-II-d	0.1450	CNSGA-II-b	0.0252
12	CNSGA-II-d	0.5761	SPEA-c	0.0057	CNSGA-II-d	0.1476	SPEA-c	0.0254
13	CNSGA-II-a	0.6122	SPEA-e	0.0062	NSGA-II-e	0.1504	SPEA-e	0.0271
14	SPEA-d	0.6285	SPEA-d	0.0065	SPEA-c	0.1685	SPEA-f	0.0280
15	CNSGA-II-b	0.6407	CNSGA-II-a	0.0066	CNSGA-II-a	0.1824	SPEA-a	0.0282
16	SPEA-e	0.6610	CNSGA-II-b	0.0068	CNSGA-II-b	0.1824	SPEA-b	0.0286
17	SPEA-a	0.8367	SPEA-b	0.0083	CNSGA-II-e	0.1920	CNSGA-II-a	0.0295
18	SPEA-b	0.8400	SPEA-a	0.0084	SPEA-d	0.2272	NSGA-e	0.0305
19	FFGA-a	1.0000	NSGA-f	0.0213	NPGA-c	0.4758	NSGA-f	0.0327
20	NPGA-a	1.0000	NSGA-e	0.0235	NPGA-e	0.4786	NPGA-e	0.0329
21	NSGA-a	1.0000	NSGA-d	0.0259	FFGA-f	0.5454	SPEA-d	0.0341
22	FFGA-b	1.0000	NPGA-f	0.0264	NPGA-f	0.5609	NPGA-f	0.0344
23	NPGA-b	1.0000	NPGA-e	0.0288	NPGA-a	0.5736	NSGA-d	0.0345
24	NSGA-b	1.0000	NPGA-d	0.0303	NSGA-d	0.5819	NSGA-c	0.0370
25	FFGA-c	1.0000	NSGA-c	0.0308	NSGA-c	0.5911	FFGA-f	0.0383
26	NPGA-c	1.0000	NSGA-b	0.0333	NPGA-d	0.5926	NSGA-b	0.0417
27	NSGA-c	1.0000	FFGA-f	0.0353	FFGA-e	0.5958	NPGA-a	0.0422
28	FFGA-d	1.0000	FFGA-e	0.0358	NSGA-a	0.5989	NPGA-d	0.0426
29	NPGA-d	1.0000	FFGA-d	0.0380	NPGA-b	0.6054	NPGA-b	0.0451
30	NSGA-d	1.0000	NPGA-c	0.0384	NSGA-f	0.6374	FFGA-e	0.0468
31	FFGA-e	1.0000	NPGA-a	0.0429	NSGA-b	0.6477	FFGA-c	0.0486
32	NPGA-e	1.0000	NPGA-b	0.0441	NSGA-e	0.6556	NSGA-a	0.0490
33	NSGA-e	1.0000	NSGA-a	0.0443	FFGA-b	0.6901	NPGA-c	0.0494
34	FFGA-f	1.0000	FFGA-c	0.0453	FFGA-d	0.7044	FFGA-d	0.0507
35	NPGA-f	1.0000	FFGA-b	0.0486	FFGA-a	0.7065	FFGA-a	0.0648
36	NSGA-f	1.0000	FFGA-a	0.0744	FFGA-c	0.7207	FFGA-b	0.0656

Tabla 7.2. Métricas Error, GD, ME y Spacing aplicadas a los resultados de las corridas.

Pocisión	Conjunto	ONVG	Conjunto	ONVGR	Conjunto	N
1	NSGA-f	626	NSGA-e	1.009124	SPEA-f	279
2	NSGA-e	553	NSGA-f	1.142336	NSGA-II-e	263
3	SPEA-f	471	SPEA-f	0.859489	NSGA-II-d	244
4	NSGA-d	445	NSGA-d	0.812044	NSGA-II-f	232
5	SPEA-d	428	SPEA-d	0.781022	CNSGA-II-f	230
6	SPEA-e	413	SPEA-e	0.753650	CNSGA-II-e	208
7	NSGA-II-f	408	NSGA-II-f	0.744526	NSGA-II-c	199
8	CNSGA-II-f	404	CNSGA-II-f	0.737226	SPEA-d	159
9	NSGA-II-e	400	NSGA-II-e	0.729927	CNSGA-II-d	156
10	CNSGA-II-e	391	CNSGA-II-e	0.713504	NSGA-II-b	154
11	NSGA-II-d	384	NSGA-II-d	0.700730	CNSGA-II-c	144
12	NPGA-f	379	NPGA-f	0.691606	SPEA-e	140
13	NSGA-II-c	378	NSGA-II-c	0.689781	SPEA-c	139
14	CNSGA-II-d	368	CNSGA-II-d	0.671533	NSGA-II-a	136
15	NSGA-II-b	344	NSGA-II-b	0.627737	CNSGA-II-b	106
16	NSGA-c	336	NSGA-c	0.613139	CNSGA-II-a	102
17	SPEA-c	327	SPEA-c	0.596715	SPEA-a	48
18	NSGA-II-a	317	NSGA-II-a	0.578467	SPEA-b	48
19	NPGA-e	313	NPGA-e	0.571168	FFGA-a	0
20	SPEA-b	300	SPEA-b	0.547445	NPGA-a	0
21	CNSGA-II-b	295	CNSGA-II-b	0.538321	NSGA-a	0
22	SPEA-a	294	SPEA-a	0.536496	FFGA-b	0
23	NSGA-b	294	NSGA-b	0.536496	NPGA-b	0
24	CNSGA-II-c	293	CNSGA-II-c	0.534672	NSGA-b	0
25	NPGA-d	293	NPGA-d	0.534672	FFGA-c	0
26	FFGA-e	280	FFGA-e	0.510949	NPGA-c	0
27	FFGA-f	267	FFGA-f	0.487226	NSGA-c	0
28	CNSGA-II-a	263	CNSGA-II-a	0.479927	FFGA-d	0
29	FFGA-d	251	FFGA-d	0.458029	NPGA-d	0
30	FFGA-c	186	FFGA-c	0.339416	NSGA-d	0
31	NPGA-c	179	NPGA-c	0.326642	FFGA-e	0
32	NSGA-a	170	NSGA-a	0.310219	NPGA-e	0
33	FFGA-b	163	FFGA-b	0.297445	NSGA-e	0
34	NPGA-a	161	NPGA-a	0.293796	FFGA-f	0
35	NPGA-b	159	NPGA-b	0.290146	NPGA-f	0
36	FFGA-a	79	FFGA-a	0.144161	NSGA-f	0

Tabla 7.3. Métricas ONVG, ONVGR y N aplicadas a los resultados de las corridas.

Posición	Conjunto	Tiempo (Segundos)
1	NPGA-a	3015.40
2	NPGA-c	3156.20
3	NSGA-a	3604.60
4	NPGA-b	3656.20
5	FFGA-c	3710.30
6	NSGA-c	4280.10
7	NSGA-b	4350.50
8	FFGA-b	4579.00
9	FFGA-d	4645.60
10	FFGA-a	4800.60
11	NPGA-d	4807.60
12	CNSGA-II-a	6102.90
13	FFGA-e	6216.70
14	CNSGA-II-b	6229.90
15	CNSGA-II-c	6294.50
16	NSGA-II-a	6314.60
17	NSGA-II-b	6344.40
18	NSGA-II-c	6801.50
19	NPGA-e	7655.30
20	NSGA-d	9390.30
21	CNSGA-II-d	10037.50
22	CNSGA-II-e	12562.70
23	FFGA-f	13754.50
24	NSGA-II-e	14850.10
25	NSGA-II-d	15453.70
26	NSGA-e	18386.30
27	NPGA-f	19665.50
28	NSGA-II-f	23516.30
29	CNSGA-II-f	26058.70
30	SPEA-a	36259.60
31	SPEA-c	37848.40
32	SPEA-b	41281.90
33	SPEA-d	44618.20
34	NSGA-f	53424.30
35	SPEA-e	57910.50
36	SPEA-f	76064.60

Tabla 7.4. Tiempo promedio utilizado para obtener cada conjunto aproximación.

	CNSGA-II-a	FFGA-a	NPGA-a	NSGA-a	NSGA-II-a	SPEA-a	Promedio
CNSGA-II-a	0.000000	1.000000	1.000000	1.000000	0.078864	0.476190	0.592509
FFGA-a	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
NPGA-a	0.000000	0.987342	0.000000	0.258824	0.000000	0.000000	0.207694
NSGA-a	0.000000	0.987342	0.447205	0.000000	0.000000	0.000000	0.239091
NSGA-II-a	0.315589	1.000000	0.993789	1.000000	0.000000	0.564626	0.645667
SPEA-a	0.079848	1.000000	0.993789	1.000000	0.053628	0.000000	0.521211
Promedio	0.065906	0.829114	0.572464	0.543137	0.022082	0.173469	

Tabla 7.5. Cobertura entre los conjuntos en las corridas a.

	CNSGA-II-b	FFGA-b	NPGA-b	NSGA-b	NSGA-II-b	SPEA-b	Promedio
CNSGA-II-b	0.000000	1.000000	0.993711	1.000000	0.075581	0.503333	0.595438
FFGA-b	0.000000	0.000000	0.000000	0.003401	0.000000	0.000000	0.000567
NPGA-b	0.000000	0.981595	0.000000	0.459184	0.000000	0.000000	0.240130
NSGA-b	0.000000	0.981595	0.471698	0.000000	0.000000	0.000000	0.242216
NSGA-II-b	0.386441	1.000000	0.987421	0.996599	0.000000	0.606667	0.662855
SPEA-b	0.088136	1.000000	0.993711	0.996599	0.043605	0.000000	0.520342
Promedio	0.079096	0.827198	0.574424	0.575964	0.019864	0.185000	

Tabla 7.6. Cobertura entre los conjuntos en las corridas b.

	CNSGA-II-c	FFGA-c	NPGA-c	NSGA-c	NSGA-II-c	SPEA-c	Promedio
CNSGA-II-c	0.000000	1.000000	1.000000	1.000000	0.150794	0.314985	0.577630
FFGA-c	0.000000	0.000000	0.000000	0.002976	0.000000	0.000000	0.000496
NPGA-c	0.000000	0.978495	0.000000	0.452381	0.000000	0.000000	0.238479
NSGA-c	0.000000	0.962366	0.329609	0.000000	0.000000	0.000000	0.215329
NSGA-II-c	0.351536	1.000000	1.000000	1.000000	0.000000	0.385321	0.622809
SPEA-c	0.225256	1.000000	1.000000	1.000000	0.185185	0.000000	0.568407
Promedio	0.096132	0.823477	0.554935	0.575893	0.055996	0.116718	

Tabla 7.7. Cobertura entre los conjuntos en las corridas c.

	CNSGA-II-d	FFGA-d	NPGA-d	NSGA-d	NSGA-II-d	SPEA-d	Promedio
CNSGA-II-d	0.000000	0.996016	1.000000	1.000000	0.091146	0.280374	0.561256
FFGA-d	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
NPGA-d	0.000000	0.972112	0.000000	0.352809	0.000000	0.000000	0.220820
NSGA-d	0.000000	0.956175	0.395904	0.000000	0.000000	0.000000	0.225347
NSGA-II-d	0.442935	0.996016	1.000000	1.000000	0.000000	0.369159	0.634685
SPEA-d	0.230978	0.996016	1.000000	1.000000	0.080729	0.000000	0.551287
Promedio	0.112319	0.819389	0.565984	0.558801	0.028646	0.108255	

Tabla 7.8. Cobertura entre los conjuntos en las corridas d.

	CNSGA-II-e	FFGA-e	NPGA-e	NSGA-e	NSGA-II-e	SPEA-e	Promedio
CNSGA-II-e	0.000000	1.000000	1.000000	1.000000	0.130000	0.438257	0.594709
FFGA-e	0.000000	0.000000	0.012780	0.003617	0.000000	0.000000	0.002733
NPGA-e	0.000000	0.982143	0.000000	0.320072	0.000000	0.000000	0.217036
NSGA-e	0.000000	0.989286	0.482428	0.000000	0.000000	0.000000	0.245286
NSGA-II-e	0.291560	1.000000	1.000000	0.996383	0.000000	0.486683	0.629104
SPEA-e	0.112532	1.000000	1.000000	1.000000	0.072500	0.000000	0.530839
Promedio	0.067349	0.828571	0.582535	0.553345	0.033750	0.154157	

Tabla 7.9. Cobertura entre los conjuntos en las corridas e.

	CNSGA-II-f	FFGA-f	NPGA-f	NSGA-f	NSGA-II-f	SPEA-f	Promedio
CNSGA-II-f	0.000000	1.000000	1.000000	1.000000	0.193627	0.242038	0.572611
FFGA-f	0.000000	0.000000	0.015831	0.012780	0.000000	0.000000	0.004768
NPGA-f	0.000000	0.943820	0.000000	0.284345	0.000000	0.000000	0.204694
NSGA-f	0.000000	0.932584	0.527704	0.000000	0.000000	0.000000	0.243381
NSGA-II-f	0.225248	1.000000	1.000000	1.000000	0.000000	0.203822	0.571512
SPEA-f	0.190594	1.000000	1.000000	1.000000	0.142157	0.000000	0.555459
Promedio	0.069307	0.812734	0.590589	0.549521	0.055964	0.074310	

Tabla 7.10. Cobertura entre los conjuntos en las corridas f.

MOEA	Corrida	Promedio	Mediana	Mínimo	Máximo	Desviación Estándar
NSGA-II	a	175	170.5	159	200	15.42725
	b	202.6	196.5	175	234	23.65117
	c	240.4	235.5	220	275	18.71541
	d	278	274	253	334	21.64871
	e	292.7	294.5	276	307	9.989439
	f	277.5	277.5	238	318	25.03442
SPEA	a	209.4	181.5	147	349	67.41282
	b	233.7	185	153	453	102.3133
	c	265.3	211.5	191	482	109.2215
	d	254.9	257.5	207	283	22.4769
	e	254.1	256	190	299	32.46348
	f	303.9	303	259	358	30.78401
CNSGA-II	a	157.1	148	127	251	34.16122
	b	175.8	163.5	155	265	33.64124
	c	192.7	194	165	210	14.92984
	d	228.7	226.5	201	264	19.51666
	e	255.3	253	232	298	19.35085
	f	289.8	299	250	323	24.23404

Tabla 7.11. Valores estadísticos para la métrica ONVG de las ejecuciones de cada corrida.

MOEA	Corrida	Promedio	Mediana	Mínimo	Máximo	Desviación Estándar
NSGA-II	a	52.8	53.5	28	72	14.04596
	b	59.4	60	35	86	15.72118
	c	76.8	76	47	103	17.83754
	d	107.9	103	95	131	12.25153
	e	117.1	116.5	80	172	27.61421
	f	100.3	94	71	144	21.50478
SPEA	a	7.3	1.5	0	34	11.44115
	b	7.4	1.5	0	34	11.52967
	c	18.4	10	0	44	19.60272
	d	33.4	27.5	3	98	30.15958
	e	27.7	22	0	67	25.408
	f	69.2	63	17	162	42.15
CNSGA-II	a	40.7	44	0	54	16.41849
	b	42.4	46.5	0	61	17.41774
	c	56.5	64	6	76	20.80732
	d	77.6	76	59	91	8.846845
	e	102.8	106	80	114	10.41153
	f	121.6	121.5	104	143	10.13465

Tabla 7.12. Valores estadísticos de la métrica N considerando las ejecuciones de cada corrida.

MOEA	Corrida	Promedio	Mediana	Mínimo	Máximo	Desviación Estándar
NSGA-II	a	6314.6	6235.5	4522	8756	1126.886
	b	6344.4	6675.5	5560	7342	677.4697
	c	6801.5	6340	5720	8280	1162.008
	d	15453.7	12824.5	7087	37522	8675.241
	e	14850.1	14591	10872	20821	3281.992
	f	23516.3	23787	17890	28423	3485.399
SPEA	a	36259.6	38241	20632	49893	9165.248
	b	41281.9	41459.5	20681	68774	14332.46
	c	37848.4	33426.5	25275	56460	11695.11
	d	44618.2	40885.5	32788	59964	9203.241
	e	57910.5	56039.5	41324	76521	11752.07
	f	76064.6	81025	52371	93689	16731.16
CNSGA-II	a	6102.9	5659.5	4576	9458	1407.376
	b	6229.9	6509.5	5499	6968	536.1993
	c	6294.5	5994	5608	7447	656.3705
	d	10037.5	9512.5	6873	15513	2508.405
	e	12562.7	12594.5	9387	15150	2453.089
	f	26058.7	26269.5	17388	33835	4212.573

Tabla 7.13. Valores estadísticos para los Tiempos en segundos de las ejecuciones de cada corrida.

Capítulo 8

Conclusiones finales y trabajos futuros

En la primera parte del presente trabajo, se ha presentado una revisión de los siguientes puntos:

1. los principios de funcionamiento de los Algoritmos Evolutivos (Capítulo 2);
2. los conceptos fundamentales de optimización multiobjetivo (Capítulo 3);
3. los algoritmos evolutivos aplicados a problemas de optimización multiobjetivo (Capítulo 4);
4. los conceptos básicos de paralelismo y la incorporación de estos en los métodos de la computación evolutiva para la resolución de problemas de optimización multiobjetivo (Capítulo 5);
5. los criterios y métricas de comparación del desempeño de los MOEAs (Capítulo 6).

Además, se propuso un modelo de paralelización basado en islas. Seis algoritmos evolutivos fueron implementados tanto en sus versiones secuenciales así como paralelas, siguiendo el modelo propuesto, para la resolución de problemas de dificultad variada:

- las funciones de prueba ZDT1 a ZDT4,
- el problema de programación óptima de bombas en estaciones de bombeo de agua potable.

Con estas implementaciones se llevaron a cabo varias corridas utilizando un número variable de procesadores. Los resultados de estas corridas fueron comparados y analizados utilizando un conjunto de métricas de desempeño.

A partir de los distintos resultados experimentales obtenidos, se han propuesto conclusiones parciales sobre la implementación de los MOEAs en la resolución de cada uno de los problemas considerados. Con estos, es posible arribar a las siguientes conclusiones finales de este trabajo:

- El diseño e implementación de algoritmos evolutivos multiobjetivo paralelos es un problema complejo. Existen varias decisiones que tomar. El rango de éstas va desde el tipo de plataforma paralela en que se realizará la implementación, hasta la determinación de diversos parámetros. Se hace necesario un mayor estudio sobre las diferentes alternativas de paralelización existentes y la determinación de parámetros adecuados de migración.

- En prácticamente todas las métricas analizadas, existe una clara diferencia entre el desempeño obtenido por las implementaciones paralelas de los MOEAs que utilizan elitismo y los que no lo utilizan. Mientras los algoritmos elitistas obtienen un conjunto de soluciones cercanas al frente Pareto óptimo con una mejor distribución de soluciones, los no elitistas apenas obtienen soluciones en dicho frente. Por ejemplo, para el problema ZDT1, como puede verse en las tablas 6.6 y 6.7, los resultados obtenidos por los algoritmos paralelos de segunda generación ocupan las posiciones 1 al 12 en todas las métricas consideradas, mientras que los de primera generación, las últimas posiciones del 13 al 24.

- Teniendo en cuenta las métricas utilizadas para medir la calidad del conjunto aproximación, se puede establecer que los algoritmos de segunda generación implementados han sido los que lograron, en general, un mejor desempeño. En particular considerando el número de soluciones verdaderamente nodominadas obtenidas. Esta diferencia se extiende en toda la gama de proble-

mas resueltos en este trabajo.

- Al aumentar el número de elementos de proceso para el modelo de paralelización utilizado, sin usar computadores paralelos, aumenta también el tiempo requerido para la finalización de una ejecución. Esto se debe principalmente a que, el tamaño promedio del conjunto solución propuesto, aumenta conforme aumenta el número de procesos que se utiliza. Este tamaño afecta al tiempo de escritura de los resultados en disco. Pero, fundamentalmente, al procedimiento de eliminación de soluciones cubiertas y reducción por clustering implementado en el procedimiento colector del modelo de paralelización utilizado. Una opción para evitar esta pérdida sería la paralelización efectiva de este procedimiento en una red de computadora, conforme fuera realizado para el problema del bombeo de agua (Capítulo 7).

- En la mayoría de los problemas analizados se ha encontrado que los algoritmos CNSGA-II y NSGA-II son los más beneficiados con la paralelización, utilizando el modelo de paralelización propuesto.

- Para ciertos problemas, en las corridas paralelas de algoritmos de primera generación ha sido posible encontrar soluciones nodominadas, aunque en un número muy reducido comparando con las implementaciones, incluso secuenciales, de los MOEAs de segunda generación.

- Para los distintos experimentos llevados a cabo, las métricas ONVGR ha sido de poca utilidad para determinar que conjunto es mejor que otro.

- Si bien los MOEAs de primera generación obtienen pocas o ninguna solución, el tiempo de ejecución de los mismos es muy inferior pues están liberados de la utilización de mecanismos para la preservación de soluciones. Estos podrían ser utilizados en forma adecuada como métodos de evolución secundaria utilizando procesos MOEAs distintos. Así, los procesos MOEAs de segunda generación se podrían encargar de proveer soluciones nodominadas a procesos MOEAs

(o pMOEAs) de primera generación, con la esperanza que estos evolucionen hacia nuevas regiones del espacio de búsqueda encontrando soluciones nodominadas distintas. Este también es un mecanismo válido para promover la formación de diversidad lateral y preservación de información genética.

- El método de reemplazo de individuos utilizado en el modelo de paralelización propuesto, evita la pérdida de soluciones nodominadas existentes en una población. Sin embargo, no garantiza el reemplazo de la "peor" solución. Se ha considerado ésta una ventaja con respecto a otros métodos pues básicamente permite el mantenimiento de puntos alejados del $P_{known}(t)$, que podría estar convergiendo en un frente Pareto óptimo local. En [104], se propone un método de reemplazo que asegura la eliminación del peor elemento. Para ello, se clasifica la población según nodominancia, para luego reemplazar entre los elementos peor clasificados, reduciendo la diversidad lateral que ha demostrado ser deseable. Con ambos métodos, cuando todos los elementos son nodominados con respecto a los migrantes o existe un único frente, la información genética existente en los migrantes es rechazada sin ser incorporada a la población genética de la isla receptora. Un método de reemplazo alternativo, podría estar basado en la idea de elitismo controlado, a fin de forzar el mantenimiento de distintos frentes.

- Existen varias métricas que tienen una utilidad relativa para medir la calidad del conjunto aproximación. Sin embargo, estas podrían integrarse en forma adecuada en un esquema de migración donde se utilice esta información para la elección de migrantes, o reemplazo. Es posible así proponer dotar al pMOEA de elementos de proceso que se especialicen en medir por ejemplo el espaciamiento de las soluciones en una isla dada y promover el desarrollo de ésta cualidad en la misma.

- Al utilizar pMOEAs, se extiende el espacio de búsqueda. La recepción de elementos prove-

nientes de distintas islas introduce información genética en forma aleatoria que es útil para la obtención de mejores soluciones. Es necesario determinar la mejor manera de incorporar esta información en las diferentes islas.

En resumen a partir de los distintos resultados presentados queda claro que la utilización de pMOEAs es adecuada para la búsqueda de soluciones en espacios de búsqueda complejos y de alta dimensionalidad. Además se ha establecido la importancia de la utilización de MOEAs basados en elitismo para la implementación de pMOEAs. Entre los MOEAs utilizados se recomienda para el desarrollo de aplicaciones paralelas la utilización del NSGA-II ya que es, en la mayoría de los problemas, el más beneficiado con la paralelización, esto a pesar que las implementaciones secuenciales del SPEA, en general, son mejores que las del NSGA-II.

A partir de las conclusiones arribadas se proponen los siguientes trabajos futuros:

- Incorporar otros enfoques al conjunto de MOEAs comparados.
- Comparar el desempeño de los distintos MOEAs utilizando distintas probabilidades de migración y número de soluciones migrantes.
- Realizar comparaciones utilizando implementaciones paralelas que utilicen distintos algoritmos evolutivos simultáneamente, formando un equipo o *team algorithm* [7].
- Desarrollar un esquema de migración que incorpore el establecimiento de preferencias del tomador de decisiones.
- Analizar el comportamiento de los pMOEAs utilizando distintos parámetros en las diferentes islas.
- Realizar un estudio experimental entre los resultados propuestos por el modelo de paralelización utilizado en [91] y el presentado en el presente trabajo.

- Desarrollar modelos paralelos para ser aplicados en sistemas multiprocesos con un único procesador.

- Realizar un estudio en mayor profundidad de las distintas métricas y como se correlacionan.

En conclusión, se puede destacar que los principales aportes del presente trabajo son:

- La presentación del estado del arte actual del desarrollo de MOEAs. Sirviendo así de base para la realización de futuras investigaciones.

- La implementación de una librería para el desarrollo rápido de pMOEAs y su utilización en varios problemas. Esta librería ha servido de base para la implementación de los distintos algoritmos y problemas presentados en el presente trabajo. Además, actualmente la librería está siendo utilizada para la resolución de otros problemas como por ejemplo el diseño óptimo de redes de computadoras.

- La propuesta de un esquema de paralelización de MOEAs basado en el modelo de islas para su implementación en sistemas paralelos con memoria distribuída.

- El análisis experimental del esquema de paralelización propuesto demostrando la efectividad de la utilización de conceptos de paralelismo en el desarrollo de MOEAs.

- La utilización de pMOEAs para la resolución del problema de la programación óptima de bombas encontrando mejores soluciones que las halladas previamente con la utilización de MOEAs secuenciales.

Glosario

Inglés	Español	Explicación o primera referencia
Multiobjective Optimization Problem	Problema de Optimización Multiobjetivo	pág. 1
Evolutionary Algorithms	Algoritmos Evolutivos	pág. 2
Evolutionary Multiobjective Optimization	Optimización Evolutiva Multiobjetivo	pág. 2
Multiobjective Evolutionary Algorithms	Algoritmos Evolutivos para Optimización Multiobjetivo	pág. 2
Parallel Multiobjective Evolutionary Algorithms	Algoritmos Evolutivos Paralelos para Optimización Multiobjetivo	pág. 2
Weak methods	Métodos débiles	pág. 6
Evolutionary Computation	Computación Evolutiva	pág. 7
Genetic Algorithm	Algoritmo Genético	pág. 7
Evolution Strategies	Estrategias de Evolución	pág. 7
Evolution Programming	Programación Evolutiva	pág. 7
Fitness	Valor de adaptación	pág. 9
Greedy algorithms	Algoritmos voraces	pág. 10
Hill-climbing algorithms	Algoritmos de ascenso de cimas	pág. 10
Best-first search	Búsqueda del mejor primero	pág. 10
No-Free-Lunch theorem	Teorema de no almuerzo gratis	pág. 12
Roulette-whell selection	Selección por ruleta ponderada	pág. 20
Tournament Selection	Selección por torneo	pág. 20
Schemata	Esquemas	pág. 22
Schema	Esquema	pág. 22
Niche Methods	Métodos de Nicho	pág. 27
Niching	Nicho	pág. 27
Fitness sharing techniques	Técnicas de distribución del valor de adaptación	pág. 27
Matting restriction	Restricción de apareamiento	pág. 29
Pareto Optimum	Óptimo de Pareto	pág. 34
Pareto Dominance	Dominancia de Pareto	pág. 34
Pareto Optimality	Optimalidad de Pareto	pág. 34
Pareto Optimal Set	Conjunto Óptimo de Pareto	pág. 34

Inglés	Español	Explicación o primera referencia
Pareto Front	Frente de Pareto	pág. 34
Decision Maker	Tomador de decisiones	pág. 39
Nondominated Sorting Genetic Algorithm	Algoritmo Genético por Clasificación de Nodominancia	pág. 53
Dummy fitness	Valor de adaptación ficticio	pág. 54
Multiple Instruction Stream Single Data Stream	Flujo múltiple de instrucciones y flujo único de datos	pág. 74
Multiple Instruction Stream Multiple Data Stream	Flujo múltiple de instrucciones y flujo múltiple de datos	pág. 74
Multiple Instruction Stream Multiple Data Stream	Flujo múltiple de instrucciones y flujo múltiple de datos	pág. 74
Single Instruction Stream Single Data Stream	Flujo único de instrucciones y flujo único de datos	pág. 74
Single Instruction Stream Multiple Data Stream	Flujo único de instrucciones y flujo múltiple de datos	pág. 74
Control Process Units	Unidad de control de procesos	pág. 75
Cluster Of Workstations - COW	Asociación de estaciones de trabajo	pág. 76
Massively Parallel Processors - MPP	Procesadores Masivamente Paralelos	pág. 76
Parallel Virtual Machine	Máquina Virtual Paralela	pág. 80
Message Passing Interface	Interfaz de paso de mensages	pág. 80
Island model	Modelo de islas	pág. 81
Diffusion model	Modelo de difusión	pág. 81
Master-slave model	Modelo maestro-esclavo	pág. 81
Local Area Network	Red de Área Local	pág. 85
Overall Non-dominated Vector Generation	Generación Total de Vectores Nodominados	pág. 98
Overall Non-dominated Vector Generation Ratio	Radio de Generación Total de Vectores Nodominados	pág. 98
Generational Distance	Distancia Generacional	pág. 99
Spacing	Espaciamiento	pág. 100
Maximum Pareto Front Error	Error máximo del frente Pareto	pág. 100
Coverage	Cobertura	pág. 100
Team algorithm	Equipo de algoritmos	pág. 170

Acrónimos

C *Coverage*

CNSGA-II *Controlled Elitist Non-dominated Sorting GA II*

COW *Cluster Of Workstations*

CPU *Control Process Unit*

DM *Decision Maker*

E *Error Ratio*

EA *Evolutionary Algorithms*

EC *Evolutionary Computation*

EMOO *Evolutionary Multi-Objetive (Multi-Criterion) Optimization*

EP *Evolution Programming*

ES *Evolution Strategies*

FFGA *Fonseca-Flemming Genetic Algorithm*

G *Generational Distance Metric*

GA *Genetic Algorithms*

GD *Generational Distance*

HLGA *Hajela Lin Genetic Algorithm*

IA *Inteligencia Artificial*

LAN *Local Area Network*

ME *Maximum Pareto Front Error*

MIMD *Multiple Instruction stream Multiple Data stream*

MISD *Multiple Instruction stream Single Data stream*

MOEA *Multiobjetive Optimization Evolutionary Algorithm*

MOGA *Multiobjective Genetic Algorithm*

MOP *Multiobjetive Optimization Problem*

MPP *Massively Parallel Processors*

N *Número de soluciones nodominadas verdaderas*

NFL *No-Free-Lunch theorem*

NPGA *Niched Pareto Genetic Algorithm*

NSGA *Nondominated Sorting Genetic Algorithm*

NSGA-II *Non-dominated Sorting GA II*

ONVG *Overall Non-dominated Vector Generation*

ONVGR *Overall Non-dominated Vector Generation Ratio*

pMOEA *Parallel Multiobjetive Optimization Evolutionary Algorithm*

PPOB *Problema de programación óptima de bombas*

PVM *Parallel Virtual Machine*

RW *Roulette-whell Selection*

S *Spacing metric*

SIMD *Single Instruction stream Multiple Data stream*

SISD *Single Instruction stream Single Data stream*

SOP *Single-objective Optimization Problem*

SPEA *Strength Pareto Evolutionary Algorithm*

TS *Tournament Selection*

VEGA *Vector Evaluated Genetic Algorithm*

Símbolos y abreviaturas

Símbolo	Significado	Definición o primera referencia
$\mathcal{S}$	Espacio de búsqueda	pág. 15
$\mathcal{A}_{\mathsf{i}}$	Conjunto de alelos para la posición i	pág. 15
$=$	Símbolo igual	pág. 15
$\times$	Producto cartesiano	pág. 15
I	Espacio de codificación	pág. 15
g	Función de codificación	pág. 15
$\mathbf{x}$	Cromosoma o vector solución de un problema	pág. 15
i	Posición de un gen en un cromosoma (*locus*)	pág. 16
$\rightarrow$	Flecha derecha	pág. 15
n	Número de genes de un cromosoma	pág. 15
x_{i}	Gen i del cromosoma x	pág. 16
f	Función monobjetivo	pág. 16
$\mathbb{R}$	Conjunto de números reales	pág. 16
l	Longitud de una cadena binaria	pág. 16
u_{i}	Valor real inferior de la variable (gen) i	pág. 17
v_{i}	Valor real superior de la variable (gen) i	pág. 17
l_{i}	Número de bits utilizados para representar la variable i en codificación binaria	pág. 17
l_{x}	$l_{\mathsf{x}} = l/n$	pág. 17
a_{j}	Valor en la posición j de una cadena binaria	pág. 17
Γ^{i}	Función de decodificación binaria para el gen i	pág. 17
$\in$	Símbolo pertenece a	pág. 17
f	Función de adaptación	pág. 18
a_{j}	El valor específico en la posición j de una cadena binaria	pág. 18
$\mathbf{x}'$	Cromosoma o vector solución de un problema	pág. 18
N	Tamaño de la población genética P	pág. 19
P_i	Elemento i en la población genética P	pág. 19
P	Población genética	pág. 19
$P(t)$	Población genética en la generación t	pág. 19

Símbolo	Significado	Definición o primera referencia
$P[i]$	Elemento i en la población genética P	pág. 19
$P[i](t)$	Elemento i en la población genética P en la generación t	pág. 19
P_j	Elemento j en la población genética P	pág. 19
$P[i]_{fitness}$	Valor de adaptabilidad del elemento $P[i]$	pág. 19
$P[i]_{chrom}$	Cromosoma de $P[i]$	pág. 19
$P[j]$	Elemento j en la población genética P	pág. 20
$P[j](t)$	Elemento j en la población genética P en la generación t	pág. 20
$<$	Símbolo menor que	pág. 20
$p_s(i)$	Probabilidad de selección por ruleta de $P[i]$	pág. 20
$\star$	Símbolo especial "no importa"	pág. 23
$\mathcal{A}_i^\star$	$\mathcal{A}_i \cup \{\star\}$	pág. 23
I^+	Conjunto de esquemas	pág. 23
H	Esquema	pág. 23
h_i	Valor en la posición i de un esquema H	pág. 23
$o(H)$	Orden de un esquema H	pág. 23
$\delta(H)$	Tamaño de definición de un esquema H	pág. 23
t	Generación actual	pág. 24
m	número de elementos en un esquema	pág. 24
$m(H,t)$	número de elementos que pertenecen al esquema H en un tiempo t	pág. 24
i_H	índice de un elemento P_j en el conjunto $\{H \wedge P(t)\}$, $i_H \in \{1,\ldots,m\}$	pág. 25
$\wedge$	Símbolo y	pág. 25
f_{i_H}	valor de adaptabilidad del elemento i_H	pág. 25
$\hat{f}()$	Valor de adaptación promedio de un conjunto de individuos	pág. 25
p_c	Probabilidad de cruzamiento	pág. 25
p_m	Probabilidad de mutación	pág. 25
$P[i]_{ncount}$	Cuenta de nicho para el individuo $P[i]$	pág. 25
σ_{share}	Umbral de diferenciabilidad es especificado por una constante	pág. 28
α	Es un parámetro constante el cual regula la forma de la función de *sharing*	pág. 28
f'_i	Valor de fitness compartido del individuo P_i	pág. 28
m_i	Cuenta de nicho para el individuo P_i	pág. 28
$d_()$	Función distancia	pág. 28
$sh()$	Función de *sharing*	pág. 28
$\mathbf{x}^\star$	Vector óptimo global	pág. 31
$f^\star$	Valor de la función f evaluada en $\mathbf{x}^\star$	pág. 31
$\mathbf{F}$	Función de optimización de k objetivos	pág. 32
k	Número de objetivos de $\mathbf{F}$	pág. 32

Símbolo	Significado	Definición o primera referencia
$\mathbf{F}'$	Función de optimización escalar derivada de **F**	pág. 32
$\mathcal{X}$	Espacio de decisión	pág. 33
$\mathcal{Y}$	Espacio objetivo	pág. 33
$\mathbf{y}$	Vector en el espacio objetivo	pág. 33
$\mathbf{g}$	Función vectorial de restricciones	pág. 33
m	Número de restricciones para un MOP general	pág. 33
$\mathcal{X}_f$	Región de factibilidad	pág. 33
$\mathcal{Y}_f$	Región de factibilidad el espacio objetivo	pág. 33
$\bigcup$	Operador unión	pág. 33
$\mathbf{0}$	Vector cuyos componentes son todos 0	pág. 33
$\mathbf{a}$	Vector solución para un problema monobjetivo	pág. 33
$\mathbf{b}$	Vector solución para un problema monobjetivo	pág. 33
$\mathbf{u}$	Vector objetivo para un problema multiobjetivo	pág. 34
$\mathbf{v}$	Vector objetivo para un problema multiobjetivo	pág. 34
v_i	Valor del componente i del vector objetivo $\mathbf{v}$	pág. 34
u_i	Valor del componente i del vector objetivo $\mathbf{v}$	pág. 34
$\succ$	Operador dominanacia	pág. 34
$\succ\succ$	Operador dominanacia de Pareto Estricta	pág. 35
$\succeq$	Operador dominanacia de Pareto débil	pág. 35
$\not\succ$	Operador nodominancia	pág. 35
$\sim$	Operador no comparable	pág. 35
Ω	Sub-conjunto de soluciones de factibles	pág. 36
$\mathbf{x}'$	Vector de decisión factible	pág. 36
$\mathcal{P}^\star$	Conjunto Pareto Óptimo Teórico	pág. 37
$\mathcal{PF}^\star$	Conjunto Pareto Óptimo Teórico	pág. 37
$\mathbf{F}_2$	Función de prueba de Schaffer	pág. 37
x	Variable númerica	pág. 37
w_i	Valor del peso para el objetivo i (Suma Ponderada)	pág. 41
T_i	Meta para el objetivo i (Programación de metas)	pág. 43
$\lvert . \rvert$	Valor absoluto	pág. 43
$\mathbf{x}_i^\star$	Solución óptima para la función objetivo i (Ordenamiento lexicográfico)	pág. 44
$f_i^\star$	Valor de la función objetivo f_i en el punto $\mathbf{x}_i^\star$ (Ordenamiento lexicográfico)	pág. 44
ε_i	Nivel permisible para el objetivo i (Métodos de restricciones ε	pág. 45
f_r	Función objetivo más relevante (Método de restricciones ε)	pág. 45
$P_{known}(t)$	Conjunto Pareto Óptimo conocido en la generación t	pág. 50
P_{known}	Conjunto Pareto Óptimo conocido la final de una corrida	pág. 50

Símbolo	Significado	Definición o primera referencia
$PF_{known}(t)$	Frente Pareto conocido en la generación t	pág. 50
PF_{known}	Frente Pareto conocido al final de una corrida	pág. 50
P_{true}	Conjunto Pareto Óptimo verdadero computable	pág. 50
PF_{true}	Frente Pareto Óptimo verdadero computable	pág. 50
$p_i^{(t)}$	Número de individuos de la población $P(t)$ que dominan a P_i	pág. 52
$P[i](t)_{rank}$	Ranking del individuo $P[i](t)$	pág. 52
$\|\| \, . \, \|\|_c$	Cardinalidad	pág. 52
df	Valor de *fitness virtual*	pág. 54
$\setminus$	Substracción de conjuntos	pág. 54
$\mathcal{F}^f$	Conjunto de individuos en el f-ésimo frente no dominado	pág. 54
$\mathcal{F}^f[i]$	Elemento *i*-ésimo del conjunto $\mathcal{F}^f$	pág. 54
sh	Valor de *sharing*	pág. 56
$\mathcal{F}^f[j]$	Elemento j-ésimo del conjunto $\mathcal{F}^f$	pág. 56
$d_{i,j}$	Distancia euclidiana entre el elemento $\mathcal{F}^f[i]$ y el elemento $\mathcal{F}_[j]^f$	pág. 56
$\mathcal{F}^f[i]_{fitness}$	Valor de adaptación del *i*-ésimo elemento del conjunto $\mathcal{F}^f$	pág. 56
$\mathcal{F}^f[j]_{objs}$	Valor del vector objetivo del *i*-ésimo elemento del conjunto $\mathcal{F}^f$	pág. 56
$\mathcal{F}^f[i]_{ncount}$	Valor de la cuenta de nicho del *i*-ésimo elemento del conjunto $\mathcal{F}^f$	pág. 56
P_{dom}	Conjunto comparación en el NPGA	pág. 57
t_{dom}	Tamaño del conjunto comparación en el NPGA	pág. 57
M	Conjunto de elementos seleccionados en el NPGA	pág. 57
P'	Población externa en el SPEA	pág. 60
N'	Tamaño máximo de la población externa	pág. 60
C	Conjunto de Clusters	pág. 61
c_1	Cluster 1	pág. 61
c_2	Cluster con distancia mínima con respecto a c_1	pág. 61
$P'[i]_{strength}$	Valor de strength del individuo $P'[i]$	pág. 61
S_i	Conjunto de soluciones dominadas por $P[i]$	pág. 64
nd_i	Número de soluciones que dominan a $P[i]$	pág. 64
$Q(t)$	Población hijo temporal utilizada en NSGA-II y CNSGA-II	pág. 67
$R(t)$	Población combinada $P(t) \cup Q(t)$	pág. 67
t_{red}	Tasa de reducción	pág. 69
η_f	Número de soluciones en el frente f permitidas para CNSGA-II	pág. 69
η_f^t	Número de soluciones existentes en el frente f	pág. 69

Símbolo	Significado	Definición o primera referencia
κ	Número de frentes existentes en $P(t)$	pág. 69
p	Número de procesos utilizados en un pMOEA	pág. 92
$N_{colector}$	Tamaño máximo de la población en el colector	pág. 92
$P_{colector}$	Población en el colector	pág. 92
n_{mig}	Número máximo de elementos a migrar	pág. 92
p_{mig}	Probabilidad de migración	pág. 92
ONVG	Métrica Generación de Total de Vectores Nodominados	pág. 98
ONVGR	Métrica Radio de Generación de Total de Vectores Nodominados	pág. 98
N	Métrica Número de Soluciones Nodominadas Verdaderas	pág. 99
E	Métrica Radio de Error	pág. 99
e_i	Valor que expresa si una solución propuesta se encuentra o no en P_{true}	pág. 99
n	Número de vectores en PF_{known}	pág. 99
$PF_{known}[i]$	Vector i-ésimo en PF_{known}	pág. 99
GD	Métrica Distancia Generacional	pág. 99
ME	Error máximo del frente Pareto	pág. 100
d_i^{min}	Distancia Euclidiana (en el espacio objetivo) entre el vector objetivo $PF_{known}[i]$ y su correspondiente más cercano en PF_{true}	pág. 100
$\overline{d^{min}}$	Valor promedio de d_i^{min}	pág. 100
S	Métrica Spacing	pág. 100
C	Métrica Cobertura	pág. 101
n_b	Número de bombas utilizadas para bombear agua desde la fuente de agua hasta un reservorio elevado	pág. 143
E_c	Costo de la energía eléctrica para una programación dada	pág. 147
C_l	Costo de la energía eléctrica en horas de costo reducido	pág. 147
C_h	Costo de la energía eléctrica en horas de costo elevado	pág. 147
t	Intervalo de tiempo considerado	pág. 147
p_t	Combinación de bombas en el intervalo t	pág. 147
$c(p_t)$	Energía eléctrica utilizada por la combinación de bombas p_t en t	pág. 147
N_s	Número total de encendidos	pág. 148

Bibliografía

[1] J. Andersson. A Survey of Multiobjective Optimization in Engineering Design. Technical Report LiTH-IKP-R-1097, Department of Mechanical Engineering, Linköping University, Linköping, Sweden, 2000.

[2] K. H. Ang y Y. Li. Multi-Objective Benchmark Studies for Evolutionary Computation. En *2001 Genetic and Evolutionary Computation Conference. Workshop Program*, págs. 393–396, San Francisco, California, Julio 2001.

[3] S. Azarm, B. J. Reynolds, y S. Narayanan. Comparison of Two Multiobjective Optimization Techniques With and Within Genetic Algorithms. En *CD-ROM Proceedings of the 25th ASME Design Automation Conference*, volume Paper No. DETC99/DAC-8584, Las Vegas, Nevada, Septiembre 1999.

[4] T. Bäck, D. B. Fogel, y Z. Michalewicz, editors. *Handbook of Evolutionary Computation*. Institute of Physics Publishing and Oxford University Press, 1997.

[5] B. Barán, J. Vallejos, R. Ramos, y U. Fernández. Multi-objective reactive power compensation. En *2001 IEEE/PES Transmission and Distribution Conference and Exposition*, volume 1, págs. 97–101. IEEE, 2001.

[6] B. Barán, J. Vallejos, R. Ramos, y U. Fernández. Reactive Power Compensation using A Multi-

objective Evolutionary Algorithm. En *IEEE Porto Power Tech Proceedings*, volume 2, págs. 6–11, Porto, Portugal, Septiembre 2001. IEEE.

[7] B. Barán, E. Kaszkurewicz, y A. Bhaya. Parallel asynchronous team algorithms: Convergence and performance analysis. *IEEE Transactions on Parallel and Distributed Systems*, 7(7):677–688, 1996.

[8] R. J. Baron y L. Higbie. *Computer architecture*. Addison-Wesley, 1992.

[9] F. Bennet, J. Koza, y M. Keane. Genetic Programming: Biologically Inspired Computation that Exhibits Creativity in Solving Non-Trivial Problems. *Simposium on AI and Scientific Creativity*, Marzo 1999.

[10] H. J. Bremermann. Optimization through evolution and recombination. En M. C. Yovits, G. T. Jacobi, y G. D. Goldstine, editors, *Self-organizing Systems*, págs. 93–106, Washington, 1962. Spartan Books.

[11] E. Cantu-Paz. A survey of parallel genetic algorithms. *Calculateurs Paralleles, Reseaux et Systemes Repartis*, 10(2):141–171, 1998.

[12] E. Cantu-Paz. Designing efficient and accurate parallel genetic algorithms. Technical Report 2108, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois, 1999.

[13] E. Cantú-Paz. A summary of research on parallel genetic algorithms. Technical Report 95007, Department of General Engineering, University of Illinois at Urbana-Champaign, Urbana, IL, 1999.

[14] V. Chankong y Y. Haimes. *Multi-objective Decision making Theory and Methodology*. Elsevier Science Publishing Co., 1983.

[15] M. Clergue y P. Collard. Dual Genetic Algorithms and Pareto Optimization. En G. D. Smith, N. C. Steele, y R. F. Albrecht, editors, *Artificial Neural Nets and Genetic Algorithms*, págs. 188–197, Norwich, UK, Abril 1997. Springer-Verlag.

[16] C. A. Coello Coello. www.lania.mx/ ccoello/EMOO/EMOObib.html.

[17] C. A. Coello Coello. An Updated Survey of GA-Based Multiobjective Optimization Techniques. Technical Report Lania-RD-98-08, Laboratorio Nacional de Informática Avanzada (LANIA), Xalapa, Veracruz, México, December 1998.

[18] C. A. Coello Coello. A comprehencive survey of evolutionary-based multiobjetive optimization techniques. *Knowledge and Information Systems. An International Journal*, 1(3):269–308, Agosto 1999.

[19] C. A. Coello Coello. A Comprehensive Survey of Evolutionary-Based Multiobjective Optimization Techniques. *Knowledge and Information Systems. An International Journal*, 1(3):269–308, Agosto 1999.

[20] C. A. Coello Coello. Constraint handling through a multiobjective optimization technique. En A. S. Wu, editor, *Proceedings of the 1999 Genetic and Evolutionary Computation Conference. Workshop Program*, págs. 117–118, Orlando, Florida, Julio 1999.

[21] C. A. Coello Coello. An Updated Survey of Evolutionary Multiobjective Optimization Techniques : State of the Art and Future Trends. En *1999 Congress on Evolutionary Computation*, volume 1, págs. 3–13, Washington, D.C., Julio 1999. IEEE Service Center.

[22] C. A. Coello Coello. An Updated Survey of GA-Based Multiobjective Optimization Techniques. *ACM Computing Surveys*, 32(2):109–143, Junio 2000.

[23] C. A. Coello Coello. A Short Tutorial on Evolutionary Multiobjective Optimization. En E. Zitzler, K. Deb, L. Thiele, C. A. Coello Coello, y D. Corne, editors, *First International Conference on Evolutionary Multi-Criterion Optimization*, págs. 21–40. Springer-Verlag. Lecture Notes in Computer Science No. 1993, 2001.

[24] C. A. Coello Coello y C. E. Mariano Romero. Evolutionary Algorithms and Multiple Objective Optimization. En M. Ehrgott y X. Gandibleux, editors, *Multiple Criteria Optimization: State of the Art Annotated Bibliographic Surveys*, págs. 277–331. Kluwer Academic Publishers, Boston, 2002.

[25] C. A. Coello Coello y G. Toscano Pulido. A Micro-Genetic Algorithm for Multiobjective Optimization. En E. Zitzler, K. Deb, L. Thiele, C. A. Coello Coello, y D. Corne, editors, *First International Conference on Evolutionary Multi-Criterion Optimization*, págs. 126–140. Springer-Verlag. Lecture Notes in Computer Science No. 1993, 2001.

[26] C. Darwin. *El origen de las especies*. Editorial Planeta-De Agostini, México, 1992.

[27] K. A. De Jong. *An analysis of the behavior of a class of genetic adaptive systems*. PhD thesis, University of Michigan, Ann Arbor, 1995. Dissertation Abstracts International 36(10), 5140B; UMI 76-9381.

[28] K. Deb. Multi-Objective Genetic Algorithms: Problem Difficulties and Construction of Test Problems. Technical Report CI-49/98, Dortmund: Department of Computer Science/LS11, University of Dortmund, Germany, 1998.

[29] K. Deb. Evolutionary algorithms for multi-criterion optimization in engineering design. En K. Miettinen, M. M. Mäkelä, P. Neittaanmäki, y J. Periaux, editors, *Evolutionary Algorithms in*

Engineering and Computer Science, págs. 135–161, Chichester, UK, 1999. John Wiley & Sons, Ltd.

[30] K. Deb. Evolutionary Algorithms for Multi-Criterion Optimization in Engineering Design. En K. Miettinen, M. M. Mäkelä, P. Neittaanmäki, y J. Periaux, editors, *Evolutionary Algorithms in Engineering and Computer Science*, chapter 8, págs. 135–161. John Wiley & Sons, Ltd, Chichester, UK, 1999.

[31] K. Deb. Multi-Objective Genetic Algorithms: Problem Difficulties and Construction of Test Problems. *Evolutionary Computation*, 7(3):205–230, Fall 1999.

[32] K. Deb. Non-Linear Goal Programming using Multi-Objective Genetic Algorithms. Technical Report CI-60/98, Dortmund: Department of Computer Science/LS11, University of Dortmund, Germany, 1999.

[33] K. Deb, S. Agrawal, A. Pratab, y T. Meyarivan. A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II. KanGAL report 200001, Indian Institute of Technology, Kanpur, India, 2000.

[34] K. Deb y T. Goel. Controlled Elitist Non-dominated Sorting Genetic Algorithms for Better Convergence. En E. Zitzler, K. Deb, L. Thiele, C. A. Coello Coello, y D. Corne, editors, *First International Conference on Evolutionary Multi-Criterion Optimization*, págs. 67–81. Springer-Verlag. Lecture Notes in Computer Science No. 1993, 2001.

[35] K. Deb y D. E. Goldberg. An investigation of niche and species formation in genetic function optimization. En J. D. Schaffer, editor, *Proceedings of the 3rd International Conference on Genetic Algorithms*, págs. 42–50, George Mason University, June 1989. Morgan Kaufmann.

[36] N. M. Duarte, A. E. Ruano, C. M. Fonseca, y P. J. Fleming. Accelerating Multi-Objective

Control System Design Using a Neuro-Genetic Approach. En *2000 Congress on Evolutionary Computation*, volume 1, págs. 392–397, Piscataway, New Jersey, Julio 2000. IEEE Service Center.

[37] S. Duarte y B. Barán. Multiobjective Network Design Optimisation Using Parallel Evolutionary Algorithms. En *XXVII Conferencia Latinoamericana de Informática CLEI-2001*, Mérida, Venezuela, 2001.

[38] M. Ehrgott y X. Gandibleux. Multiple Objective Combinatorial Optimization - A Tutorial, 2002. Presentación en Fourth International Conference on Multi-Objective Programming and Goal Programming MOPGP'02, http://www.esc.auckland.ac.nz/People/Staff/Matthias/.

[39] A. G. et al. *PVM:Paralell Virtual machine - A user's guide and Tutorial for Networked parallel Computing*. M.I.T. press, Cambridge, MA, 1994.

[40] R. Flynn y P. D. Sherman. Multicriteria Optimization of Aircraft Panels: Determining Viable Genetic Algorithm Configurations. *International Journal of Intelligent Systems*, 10:987–999, 1995.

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

[42] C. M. Fonseca y P. J. Fleming. Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization. En S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, págs. 416–423, San Mateo, California, 1993. University of Illinois at Urbana-Champaign, Morgan Kauffman Publishers.

[43] C. M. Fonseca y P. J. Fleming. Multiobjective Genetic Algorithms. En *IEE Colloquium on Genetic Algorithms for Control Systems Engineering*, págs. 6/1–6/5. IEE, 1993.

[44] C. M. Fonseca y P. J. Fleming. An Overview of Evolutionary Algorithms in Multiobjective Optimization. Technical report, Department of Automatic Control and Systems Engineering, University of Sheffield, Sheffield, U. K., 1994.

[45] C. M. Fonseca y P. J. Fleming. An Overview of Evolutionary Algorithms in Multiobjective Optimization. *Evolutionary Computation*, 3(1):1–16, Spring 1995.

[46] I. Foster. *Designing and Building Parallel Programs*. Addison-Wesley Publishing Company, Reading, MA, 1995.

[47] M. P. Fourman. Compaction of Symbolic Layout using Genetic Algorithms. En *Genetic Algorithms and their Applications: Proceedings of the First International Conference on Genetic Algorithms*, págs. 141–153. Lawrence Erlbaum, 1985.

[48] M. Ginsberg. *Essentials of Artificial Intelligence*. Morgan Kaufman Publishers, San Francisco, USA, 1993.

[49] D. E. Goldberg. Genetic and evolutionary algorithms come of age. *Communications of the ACM*, 37(3):113–119, Marzo 1994.

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

[51] J. H. Holland. Outline for a logical theory of adaptive systems. *Journal of the ACM*, 9(3):297–314, Julio 1962.

[52] J. H. Holland. *Adaptation in natural artificial systems*. University of Michigan Press, Ann Arbor, 1975.

[53] J. Horn. *The Nature of Niching: Genetic Algorithms and the Evolution of Optimal, Cooperative Populations*. PhD thesis, University of Illinois at Urbana Champaign, Urbana, Illinois, 1997.

[54] J. Horn y N. Nafpliotis. Multiobjective Optimization using the Niched Pareto Genetic Algorithm. Technical Report IlliGAl Report 93005, University of Illinois at Urbana-Champaign, Urbana, Illinois, USA, 1993.

[55] J. Horn, N. Nafpliotis, y D. E. Goldberg. A Niched Pareto Genetic Algorithm for Multiobjective Optimization. En *Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, volume 1, págs. 82–87, Piscataway, New Jersey, Junio 1994. IEEE Service Center.

[56] L. C. Jain y N. M. Martin. *Fusion of Neural Networks, Fuzzy Sets, and Genetic Algorithms : Industrial Applications*. International Series on Computational Intelligence, USA, 1999.

[57] S. Khuri, T. Bäck, y J. Heitkötter. The zero/one multiple knapsack problem and genetic algorithms. En *Proceedings of the 1994 ACM Symposium on Applied Computing*, págs. 188–193, New York, 1994. ASME Press.

[58] S. Khuri y T. Bäck. An evolutionary heuristic for the maximum independent set problem. En *IEEECEP: Proceedings of The IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, 1994.

[59] K. E. Lansey y K. Awumah. Optimal pump operations considering pump switches. *Water Resources Planning and Management Journal*, 1(120), 1994.

[60] M. Laumanns, L. Thiele, E. Zitzler, E. Welzl, y K. Deb. Running Time Analysis of Multiobjective Evolutionary Algorithms on a Simple Discrete Optimization Problem. En J. J. Merelo Guervós, P. Adamidis, H.-G. Beyer, J.-L. F.-V. nas, y H.-P. Schwefel, editors, *Parallel Problem*

Solving from Nature—PPSN VII, págs. 44–53, Granada, Spain, Septiembre 2002. Springer-Verlag. Lecture Notes in Computer Science No. 2439.

[61] D. Mackle, A. Savic, y G. A. Walters. Application of genetic algorithms to pump scheduling for water supply. En *GALESIA 95*, Londres, UK, 1995.

[62] S. W. Mahfoud. *Niching methods for genetic algorithms*. PhD thesis, University of Michigan, Urbana, IL, USA, 1995.

[63] S. W. Mahfoud. Niching methods. En T. Bäck, D. B. Fogel, y Z. Michalewicz, editors, *Handbook of Evolutionary Computation*, págs. C6.1:1–4. Institute of Physics Publishing and Oxford University Press, Bristol, New York, 1997.

[64] K. Miettien. Some methods for nonlinear multi-objetive optimization. En E. Zitzler, K. Deb, L. Thiele, C. A. Coello Coello, y D. Corne, editors, *First International Conference on Evolutionary Multi-Criterion Optimization*. Springer-Verlag. Lecture Notes in Computer Science No. 1993, 2001.

[65] L. E. Ormsbee y K. E. Lansey. Optimal control of water supply pumping systems. *Water Resources Planning and Management Journal*, 1994.

[66] P. Pacheco. *Parallel Programming with MPI*. Morgan Kaufmann, San Francisco, CA, 1997.

[67] V. Pareto. *Cours D'Economie Politique I y II*. F. Rouge, Lausanne, 1896.

[68] R. Purshouse y P. Fleming. The Multi-Objective Genetic Algorithm Applied to Benchmark Problems—An Analysis. Technical Report 796, Department of Automatic Control and Systems Engineering, University of Sheffield, Sheffield, UK, Agosto 2001.

[69] I. Rechenberg. Cybernetic solution path of an experimental problem. Technical report, Royal Air Force Establishment, Farnborough, UK, 1965.

[70] G. Rudolph. Convergence analysis of canocical genetic algorithms. *IEEE Transactions on Neural Networks*, 5(1):96–101, 1993.

[71] G. Rudolph. On a Multi-Objective Evolutionary Algorithm and Its Convergence to the Pareto Set. En *Proceedings of the 5th IEEE Conference on Evolutionary Computation*, págs. 511–516, Piscataway, New Jersey, 1998. IEEE Press.

[72] S. Russell y P. Norvig. *Artificial Intelligence a Modern Approach*. AI. Prentice–Hall, 1995.

[73] B. Sareni y L. Krähenbühl. Fitness sharing and methods revisited. *IEEE Transactions on Evolutionary Computation*, 2(3), Septiembre 1998.

[74] D. Sasaki, M. Morikawa, y S. Obayashi. Aerodynamic shape optimization of supersonic wings by adaptive range multiobjective genetic algorithms. En E. Zitzler, K. Deb, L. Thiele, C. A. Coello Coello, y D. Corne, editors, *First International Conference on Evolutionary Multi-Criterion Optimization*, págs. 639–652, Germany, Marzo 2001. Springer-Verlag. Lecture Notes in Computer Science No. 1993.

[75] D. A. Savic, G. A. Walters, y M. Schwab. Multiobjective Genetic Algorithms for Pump Scheduling in Water Supply. En *AISB International Workshop on Evolutionary Computing. Lecture Notes in Computer Science 1305*, págs. 227–236, Berlin, April 1997. Springer-Verlag.

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

[77] J. D. Schaffer y J. J. Grefenstette. Multiobjective Learning via Genetic Algorithms. En *Pro-*

ceedings of the 9th International Joint Conference on Artificial Intelligence (IJCAI-85), págs. 593–595, Los Angeles, California, 1985. AAAI.

[78] M. Schwab, D. A. Savic, y G. A. Walters. Multi-Objective Genetic Algorithm for Pump Scheduling in Water Supply Systems. Technical Report 96/02, Centre For Systems And Control Engineering, School of Engineering, University of Exeter, Exeter, United Kingdom, 1996.

[79] H. P. Schwefel. *Kybernetische Evolution als Strategie der Experimentellen Forschung in der Strömungstechnik*. Hermann Föttinger Institut für Strömungstechnik, Berlin, 1965.

[80] A. Sotelo y B. Barán. Pumping cost optimization in water supply systems using a multiobjective evolutionary combined algorithm. En *15 Conferencia Chilena de Ingeniería Hidráulica*, Concepción, Chile, 2001.

[81] N. Srinivas y K. Deb. Multiobjective optimization using nondominated sorting in genetic algorithms. Technical report, Department of Mechanical Engineering, Indian Institute of Technology, Kanpur, India, 1993.

[82] N. Srinivas y K. Deb. Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms. *Evolutionary Computation*, 2(3):221–248, Fall 1994.

[83] N. Srinivas y K. Deb. Comparative study of vector evaluated GA and NSGA applied to multiobjective optimization. En P. K. Roy y S. D. Mehta, editors, *Proceedings of the Symposium on Genetic Algorithms*, págs. 83–90, 1995.

[84] A. S. Tanenbaum. *Organización de computadoras, un enfoque estructurado, 4ta Edición*. Prentice Hall, 2000.

[85] A. Turing. Computing machinery and intelligence. En G. F. Luger, editor,

Computation and Intelligence:Collected Readings. AAAI Press/The MIT Press, 1995. http://www.abelard.org/turpap/turpap.htm.

[86] D. A. van Veldhuizen. *Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovations*. PhD thesis, Department of Electrical and Computer Engineering. Graduate School of Engineering. Air Force Institute of Technology, Wright-Patterson AFB, Ohio, Mayo 1999.

[87] D. A. van Veldhuizen y G. B. Lamont. On Measuring Multiobjective Evolutionary Algorithm Performance. En *2000 Congress on Evolutionary Computation*, volume 1, págs. 204–211, Piscataway, New Jersey, Julio 2000. IEEE Service Center.

[88] D. A. van Veldhuizen, J. B. Zydallis, y G. B. Lamont. Issues in Parallelizing Multiobjective Evolutionary Algorithms for Real World Applications. En *Proceedings of the 17th ACM Symposium on Applied Computing*, págs. 595–602, Madrid, Spain, 2002. ACM Press.

[89] D. A. van Veldhuizen, J. B. Zydallis, y G. B. Lamont. Considerations in Engineering Parallel Multiobjective Evolutionary Algorithms. *IEEE Transactions on Evolutionary Computation*, 7(2):144–173, April 2003.

[90] C. von Lücken y B. Barán. Multi-objective evolutionary algorithms experimental comparison. En *Conferencia Internacional de Tecnologías y Aplicaciones Informáticas*, Asunción, Paraguay, 2001.

[91] C. von Lücken, B. Barán, y A. Sotelo. Pump schedulling optimisation using parallel multiobjective evolutionary algorithms. En *XXVII Conferencia Latinoamericana de Informática CLEI-2003*, La Paz, Bolivia, 2003.

[92] C. von Lücken, A. Sotelo, y B. Barán. Multiobjective evolutionary algorithms in pump scheduling optimisation. En B. Topping y Z. Bittnar, editors, *Proceedings of the Third International Conference on Engineering Computational Technology*, Stirling, United Kingdom, 2002. Civil-Comp Press.

[93] D. Whitley. A genetic algorithm tutorial. Technical Report CS-93-103, Colorado State University, 1993.

[94] D. H. Wolpert y W. G. Macready. No free lunch theorems for search. Technical Report SFI-TR-95-02-010, Santa Fe Institute, Santa Fe, NM, 1995.

[95] D. H. Wolpert y W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, April 1997.

[96] E. Zitzler. *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*. PhD thesis, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, Noviembre 1999.

[97] E. Zitzler, K. Deb, y L. Thiele. Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. Technical Report 70, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zurich, Gloriastrasse 35, CH-8092 Zurich, Switzerland, December 1999.

[98] E. Zitzler, K. Deb, y L. Thiele. Comparison of Multiobjective Evolutionary Algorithms on Test Functions of Different Difficulty. En A. S. Wu, editor, *Proceedings of the 1999 Genetic and Evolutionary Computation Conference. Workshop Program*, págs. 121–122, Orlando, Florida, Julio 1999.

[99] E. Zitzler, K. Deb, y L. Thiele. Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. *Evolutionary Computation*, 8(2):173–195, Summer 2000.

[100] E. Zitzler, K. Deb, L. Thiele, C. A. Coello Coello, y D. Cornde, editors. *Proceedings of the First International conference on EMOO, 2001*, Berlin, Germany, Marzo 2001. Springer-Verlag.

[101] E. Zitzler, M. Laumanns, y L. 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, Mayo 2001.

[102] E. Zitzler y L. Thiele. An Evolutionary Algorithm for Multiobjective Optimization: The Strength Pareto Approach. Technical Report 43, Computer Engineering and Communication Networks Lab (TIK), Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, Mayo 1998.

[103] E. Zitzler y L. Thiele. Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, Noviembre 1999.

[104] J. B. Zydallis y G. B. Lamont. Solving of Discrete Multiobjective Problems Using an Evolutionary Algorithm with a Repair Mechanism. En *Proceedings of the IEEE 2001 Midwest Symposium on Circuits and Systems*, volume 1, págs. 470–473. IEEE, 2001.

Índice alfabético