UNIVERSIDAD NACIONAL DE ASUNCIÓN

FACULTAD POLITÉCNICA

OPTIMIZACIÓN DE ENJAMBRE DE PARTÍCULAS APLICADA AL PROBLEMA DEL CAJERO VIAJANTE BI–OBJETIVO

JOAQUÍN QUINTO LIMA MOLINARI

2007

UNIVERSIDAD NACIONAL DE ASUNCIÓN

FACULTAD POLITÉCNICA

OPTIMIZACIÓN DE ENJAMBRE DE PARTÍCULAS APLICADA AL PROBLEMA DEL CAJERO VIAJANTE BI–OBJETIVO

JOAQUÍN QUINTO LIMA MOLINARI

PROYECTO DE TRABAJO DE GRADO PRESENTADO EN CONFORMIDAD A LOS REQUISITOS PARA OBTENER EL GRADO DE INGENIERO EN INFORMÁTICA.

ASESOR: Prof. D. Sc. BENJAMÍN BARÁN CEGLA

SAN LORENZO – 2007

Resumen

Muchas aplicaciones prácticas requieren de la resolución de *Problemas de Optimización Combinatoria* (COP – *Combinatorial Optimization Problems*). Estos problemas son resueltos encontrando un ordenamiento, selección, agrupación, combinación o permutación de un número finito de elementos discretos, operaciones o pasos de manera a obtener el mayor provecho posible en el caso práctico relacionado.

Por otra parte, ciertos problemas también involucran la identificación de soluciones considerando simultáneamente múltiples criterios de evaluación. A estos problemas se los denomina *Problema de Optimización Multiobjetivo* (MOP – *Multiobjective Optimization Problem*) y se resuelven encontrando el conjunto de soluciones que optimicen a la vez las evaluaciones de todos los objetivos.

La mayoría de los problemas de estos tipos, COPs y MOPs, tienen un espacio de búsqueda muy grande y/o complejo, lo cual hace que las técnicas de resolución tradicionales no resulten aptas para ser aplicadas. Por lo tanto, se hace evidente la necesidad de nuevas técnicas que permitan el tratamiento efectivo de estos problemas.

Los *Algoritmos Evolutivos* (EA – *Evolutionary Algorithms*) y la *Optimización de Colonia de Hormigas* (ACO – *Ant Colony Optimization*) son ejemplos de tales técnicas alternativas y en la literatura se pueden encontrar varios trabajos que reportan el éxito de su aplicación en la resolución de *Problemas de Optimización Combinatoria de Múltiples Objetivos* (MOCOP – *MultiObjective Combinatorial Optimization Problems*).

Este trabajo presenta la aplicación de la metaheurística de *Optimización de Enjambre de Partículas* (PSO – *Particle Swarm Optimization*) para resolver el Problema del Cajero Viajante (TSP – *Traveling Salesman Problem*) bi-objetivo, cuyo objetivo principal es probar empíricamente la efectividad de la aplicación de la metaheurística PSO en la resolución de MOCOPs. Para ello se implementaron enfoques de la metaheurística PSO para la resolución de MOPs (MOPSO – *MultiObjective Particle Swarm Optimization*) y una adaptación del PSO para el tratamiento del TSP.

La aplicación es validada comparando las soluciones obtenidas por implementaciones basadas en la metaheurística PSO contra las soluciones proporcionadas por implementaciones basadas en la metaheurística ACO, utilizando como referencia las soluciones encontradas por enfoques basados en técnicas tradicionales de resolución de MOPs y del TSP mono-objetivo.

Los resultados de las comparaciones indican que el uso de las técnicas MOPSO junto con la adaptación del PSO para el TSP constituyen una buena alternativa para la resolución del TSP bi-objetivo, demostrándose empíricamente que los conceptos de la metaheurística PSO también pueden ser aplicados en la resolución de MOCOPs.

Abstract

Many practical applications require the resolution of *Combinatorial Optimization Problems* (COP). These problems are solved by finding an order, selection, grouping, combination or permutation of a finite number of discreet elements, operations or steps to obtain the greater possible benefit in the related practical application.

On the other hand, certain problems also involve the identification of solutions considering simultaneously multiple criteria of evaluation. These problems are denominates *Multiobjective Optimization Problem* (MOP) and are solved by finding the set of solutions that simultaneously optimize the evaluations of all objectives.

Most problems of these types, COPs and MOPs, have a very great and/or complex search's space, which causes that traditional resolutions techniques turn out nonapt to be applied. Therefore, it becomes evident the necessity of new techniques that allow the effective treatment of these problems.

Evolutionary Algorithms (EA) and *Ant Colony Optimization* (ACO) are examples of such techniques. The literature reports it success in several applications for resolutions of *MultiObjective Combinatorial Optimization Problems* (MOCOP).

This work presents an application of *Particle Swarm Optimization* (PSO) metaheuristic to solve the bi-objective *Traveling Salesman Problem* (TSP). The main objective is to prove empirically the effectiveness of PSO concepts application in MOCOPs resolutions. For this, proposals of PSO metaheuristic for MOPs (MOPSO) and an adaptation of PSO for the TSP were implemented.

The application is validated by comparing the solutions obtained by implementations based on PSO metaheuristic against the solutions provided by implementations based on ACO metaheuristic, taking like references the solutions found by approaches based on traditional resolutions techniques of MOPs and the single objective TSP.

The comparison results indicate that the use of techniques as MOPSO along with the adaptation of PSO for the TSP constitutes a good alternative for the bi-objective TSP resolution, which demonstrated empirically that PSO concepts can also be applied in MOCOPs resolution.

Índice General

Índice de Figuras

Índice de Tablas

Lista de Siglas y Acrónimos

ACM	*Association for Computing Machinery*
ACO	*Ant Colony Optimization*
ACS	*Ant Colony System*
clMOPSO	Método MOPSO de Coello y Lechuga
CO	*Combinatorial Optimization*
COP	*Combinatorial Optimization Problem*
CP	Conjunto Pareto
CS	*Current Solution Set*
D-TPLS	*Double Two Phase Local Search*
EA	*Evolutionary Algorithm*
EP	*Evolutionary Programming*
ES	*Evolution Strategies*
FP	Frente Pareto
fsMOPSO	Método MOPSO de Fieldsend y Singh
GA	*Genetic Algorithm*
GP	*Genetic Programming*
heMOPSO	Método MOPSO de Hu y Eberhart
IEEE	*Institute of Electrical and Electronics Engineers*
LIFO	*Last in, Last Out*
M3AS	*Multiobjective Max-Min Ant System*
mcMOPSO	Método MOPSO de Moore y Chapman
MOACO	*MultiObjective Ant Colony Optimization*
MOACS	*MultiObjective Ant Colony System*
MOCO	*MultiObjective Combinatorial Optimization*
MOCOP	*MultiObjective Combinatorial Optimization Problem*
MOGLS	*MultiObjective Genetic Local Search*
MOP	*Multiobjective Optimization Problem*
MOPSO	*MultiObjective Particle Swarm Optimization*
mtMOPSO	Método MOPSO de Mostaghim y Teich
NASA	*National Aeronautics and Space Administration*
NP	*Non-polynomic Deterministic Time*

NPGA	*Niched Pareto Genetic Algorithm*
NSGA	*Nondominated Sorting Genetic Algorithm*
NSGA2	*Nondominated Sorting Genetic Algorithm II*
PACO	*Pareto Ant Colony Optimization*
PAES	*Pareto Archived Evolutionary Strategy*
PCB	*Printed Circuit Board*s
PD-TPLS	*Pareto Double Two Phase Local Search*
PE	*Potential Efficient Solutions*
PLS	*Pareto Local Search*
PSO	*Particle Swarm Optimization*
QAP	*Quadratic Assignment Problem*
SPEA	*Strength Pareto Evolutionary Algorithm*
SPEA2	*Strength Pareto Evolutionary Algorithm 2*
TA	*Team Algorithm*
TP	*Temporary Population*
TPLS	*Two Phase Local Search*
TSP	*Traveling Salesman Problem*
TSPLIB	Librería de problemas TSP
VRP	*Vehicle Routing Problem*

Capítulo 1
Introducción

1.1 Nociones Preliminares

Una clase importante de problemas es conocida como *Problema de Optimización Combinatoria* (COP – *Combinatorial Optimization Problem*) [Wikipedia1, Hansen98]. Los problemas de esta clase buscan optimizar uno o más criterios u objetivos mediante la construcción de soluciones que consisten en una selección, ordenación, agrupación, combinación o permutación de un número finito de elementos discretos, operaciones o pasos según sea el caso. Por lo general, la mayoría de estos problemas son clasificados como NP-Duros y/o NP-Completos [Sahni1976], es decir que, aún no existen algoritmos exactos con tazas de tiempo de ejecución polinomial que resuelvan estos problemas utilizando máquinas de computo determinísticas.

Otra clase de problemas es la denominada clase de *Problemas de Optimización Multiobjetivo* (MOP – *Multiobjective Optimization Problem*) [Coello99, Veldhuizen99, Zitzler00]. Los problemas de dicha clase pretenden en encontrar soluciones que optimicen simultáneamente dos o más criterios de evaluación que a menudo son contradictorios e inconmensurables entre si. Por lo general, la mayoría de los MOPs no poseen una única solución que permita obtener la mejor evaluación en cada objetivo. Por el contrario, para cada MOP existe un conjunto de soluciones óptimas dentro del cual ninguna solución puede ser considerada mejor a las demás.

Debido a que muchos problemas de la ingeniería, la industria, los negocios y la ciencia son formulados como COPs [Yu98], que en ocasiones requieren de la búsqueda de soluciones que satisfagan simultáneamente dos o más criterios de evaluación (MOCOP – *MultiObjective Combinatorial Optimization Problem*), surge la necesidad de técnicas que permitan resolver estos problemas en un tiempo razonable y de forma eficaz. En algunos casos, estos problemas presentan cierta dificultad relacionada a la cantidad de soluciones a evaluar y/o a la complejidad de su espacio de resolución que imposibilita a las técnicas tradicionales a obtener soluciones óptimas [Lücken03].

Para resolver los MOPs en donde las técnicas tradicionales fallan aparecen los métodos de Inteligencia Artificial. Estos métodos utilizan una especificación de alto nivel del problema a resolver y una estrategia de búsqueda inspirada en la naturaleza que les permiten descubrir las soluciones óptimas a estos casos de manera eficiente. Ejemplos de tales técnicas son los

Algoritmos Evolutivos (EA – *Evolutionary Algorithms*), inspirados en la teoría de la evolución Darwiniana, y los algoritmos de *Optimización de Colonias de Hormigas* (ACO – *Ant Colony Optimization*), motivados por el comportamiento de las hormigas durante la búsqueda de alimentos, sobre los cuales se pueden encontrar en la literatura numerosos trabajos que reportan el éxito de su aplicación.

El *Problema del Cajero Viajante* (TSP – *Traveling Salesman Problem*), que en su versión multiobjetivo cae dentro de la tipificación de MOCOP, es posiblemente uno de los COPs más utilizados para probar la efectividad de las técnicas de optimización en la resolución de problemas de este tipo. Esto se debe a la facilidad de su comprensión y a que presenta una complejidad similar a la mayoría de los COPs más difíciles de resolver.

Por otra parte, la metaheurística de *Optimización de Enjambre de Partículas* (PSO – *Particle Swarm Optimization*) [Kennedy95] es una técnica estocástica de resolución de problemas continuos de optimización, que está basada en el comportamiento social observado en ciertas agrupaciones de aves, peces e insectos y sobre la cual puede encontrarse en la literatura varios trabajos que presentan resultados exitosos de su aplicación.

1.2 Alcance y Objetivo

El presente trabajo estudia la aplicación de los conceptos de la metaheurística PSO en la resolución del TSP bi-objetivo, validando esta propuesta al comparar los resultados arrojados por implementaciones de enfoques de resolución de MOPs basados en la metaheurística PSO (MOPSO – *MultiObjective Particle Swarm Optimization*) contra los resultados obtenidos por implementaciones basadas en la metaheurística ACO para la resolución de MOPs (MOACO – *MultiObjective Ant Colony Optimization*), utilizándose como referencia soluciones obtenidas mediante métodos basados en técnicas tradicionales de resolución de MOPs y del TSP mono-objetivo.

El objetivo principal es probar empíricamente la validez de la aplicación de los conceptos de la metaheurística PSO en la resolución de MOCOPs a través la resolución del TSP bi-objetivo.

A la vez, este trabajo puede ser considerado como una extensión del trabajo de Secrest [Secrest01], en el cual se realiza una adaptación de la metaheurística PSO para resolver el TSP mono-objetivo. Así mismo, también puede ser visto como uno de los primeros trabajos en el que se comparan entre si varias propuestas de métodos MOPSOs y MOACOs, utilizando

como referencia resultados obtenidos a través de métodos basados en técnicas tradicionales de resolución de MOPs y del TSP mono-objetivo.

1.3 Organización del Trabajo

El resto del trabajo se encuentra organizado como sigue:

En el capítulo 2 se presentan una introducción a los MOPs, su formulación, el tratamiento de sus soluciones y una clasificación de los métodos de resolución de MOPs. También se presentan algunos métodos tradicionales y métodos basados en EA para la resolución de MOPs.

En el capítulo 3 se introduce la definición formal del TSP, se citan algunas de sus aplicaciones prácticas y se presenta un breve resumen de las bases de la metaheurística ACO, la cual es comúnmente utilizada para la resolución de COPs. También se presenta una descripción de los métodos MOACOs utilizados en este trabajo y de los métodos de resolución del TSP multiobjetivo basados en técnicas tradicionales de resolución de MOPs y del TSP mono-objetivo.

El capítulo 4 trata de los fundamentos de la metaheurística PSO, su implementación computacional para la resolución de problemas mono-objetivos y multiobjetivos, y del establecimiento de los parámetros para los algoritmos PSO. En el mismo capítulo, se presenta un resumen de algunos métodos MOPSO encontrados en la literatura que fueron implementados para la realización de este trabajo y la adaptación aplicada a la metaheurística PSO para resolver el TSP.

En el capítulo 5 se presenta los resultados de las comparaciones entre las soluciones obtenidas por las implementaciones de los métodos MOPSOs y MOACOs en la resolución de cuatro instancias del TSP bi-objetivo, utilizando como referencia resultados obtenidos a través de métodos basados en técnicas tradicionales.

Finalmente, en el capítulo 6 se presentan las conclusiones finales de este trabajo y se proponen algunos puntos que pueden ser considerados como posibles trabajos futuros.

Capítulo 2

Problemas de Optimización Multiobjetivo

2.1 Introducción

En diversas áreas, tales como la ciencia, la ingeniería, la economía y la industria, muchos problemas necesitan ser resueltos teniendo en cuenta varios objetivos de evaluación, los cuales a menudo son contradictorios e inconmensurables entre si. Es decir que, cuando se mejora la evaluación en un objetivo inevitablemente se desmejora la evaluación de al menos uno de los objetivos restantes y además estos objetivos pueden cuantificarse en magnitudes diferentes, tales como segundos, kilómetros, dinero, etc. A esta clase de problemas se denomina *Problema de Optimización Multiobjetivo* (MOP - *Multiobjective Optimization Problem*) [Coello99, Veldhuizen99, Zitzler00].

Los MOPs con dos o más objetivos en conflicto tienen la característica de que para ellos no existe una única solución que permita obtener la mejor evaluación en todos los objetivos, sino que en estos casos se tiene un conjunto de soluciones igualmente factibles, en donde cada una de las soluciones puede ser considerada como óptima para el problema bajo determinadas circunstancias y donde todas ellas representan las mejores combinaciones de las evaluaciones posibles en todos los objetivos [Coello99, Veldhuizen99, Zitzler00].

Un ejemplo sencillo de un MOP es la elección del diseño para un vehículo en donde deben considerarse a los siguientes objetivos: (i) *minimizar el costo de fabricación* y (ii) *maximizar la seguridad del vehículo*. Evidentemente estos objetivos son contradictorios entre si, ya que al agregar por ejemplo más dispositivos o aparatos de protección a un diseño específico, se está incrementando tanto la seguridad como el costo de fabricación del vehículo según dicho diseño.

De esta forma, es posible encontrar varios diseños para el vehículo que pueden ser considerados como soluciones óptimas al problema. Cada uno con su correspondiente par de valores que indican el costo de fabricación y la seguridad del vehículo (evaluación de los objetivos), de forma que al compararlos de a dos todos contra todos, encontramos siempre que un diseño resulta mejor en un objetivo mientras que el otro diseño resulta mejor en el otro objeti-

vo. En consecuencia, en ausencia de un método de comparación basado en ambos criterios, no se puede afirmar que uno de los diseños sea el *mejor* entre todos, por lo que todos resultan ser *igualmente factibles* u *óptimos*.

En las siguientes secciones de este capítulo se presentan: la formulación matemática de los MOPs, los conceptos de Optimalidad Pareto, una clasificación de los métodos de resolución de MOPs, un resumen de las técnicas tradicionales de resolución de MOPs, una introducción a la teoría de los Algoritmos Evolutivos (EA – *Evolutionary Algorithm*) y un resumen de las algunas técnicas basadas en la teoría de EA para la resolución de MOPs.

2.2 Formulación Multiobjetivo

Definición 2.1 (MOP): De forma general, un MOP se expresa formalmente como [Coello99, Veldhuizen99, Zitzler00]:

$$\textit{Optimizar} \quad \vec{y} = \vec{f}(\vec{x}) = [f_1(\vec{x}), f_2(\vec{x}), \ldots, f_M(\vec{x})] \tag{2.1}$$

sujeto a:

$$\vec{g}(\vec{x}) = [g_1(\vec{x}), g_2(\vec{x}), \ldots, g_J(\vec{x})] \geq 0, \tag{2.2}$$

$$\vec{h}(\vec{x}) = [h_1(\vec{x}), h_2(\vec{x}), \ldots, h_K(\vec{x})] = 0, \tag{2.3}$$

donde $\vec{x} = [x_1, x_2, ..., x_N]$ es el *vector de decisión* con los valores para las N variables de decisión del problema, $\vec{y} = [y_1, y_2, \ldots, y_M]$ es el *vector solución* con las evaluaciones de las M funciones objetivos $f_i(\vec{x})$, y las funciones $g(\vec{x})$ y $h(\vec{x})$ representan respectivamente a las J restricciones de desigualdad y K restricciones de igualdad sobre el espacio de las variables de decisión. Por *Optimizar* se entiende la *minimización* o *maximización* de cada una de las M funciones objetivos.

Evidentemente, cualquier objetivo de maximización puede convertirse en un objetivo de minimización multiplicando su respectiva ecuación por -1 y viceversa.

Definición 2.2 (espacio de decisión factible): El *espacio de decisión factible* es el conjunto de todos los vectores de decisión $\vec{x}$ en el espacio $\Re^N$ que atienden a las restricciones de desigualdad $\vec{g}(\vec{x})$ e igualdad $\vec{h}(\vec{x})$, que se denota como:

$$\Omega = \{ \vec{x} \mid \vec{x} \in \Re^N \wedge \vec{g}(\vec{x}) \geq 0 \wedge \vec{h}(\vec{x}) = 0 \}.$$

Definición 2.3 (espacio solución posible): El *espacio solución posible* es el conjunto de los vectores solución $\vec{y}$ que constituyen la imagen de Ω a través $\vec{f}(\vec{x})$ en el espacio solución o de las funciones objetivo $\Re^M$, denotado como:

$$\Omega_0 = \{ \vec{y} \mid \vec{y} \in \Re^M \wedge \vec{y} = \vec{f}(\vec{x}) \; \forall \vec{x} \in \Omega \}.$$

En la figura 2.1 puede observase una ilustración de estos conceptos para el caso de un MOP con dos objetivos de evaluación y dos variables de decisión.

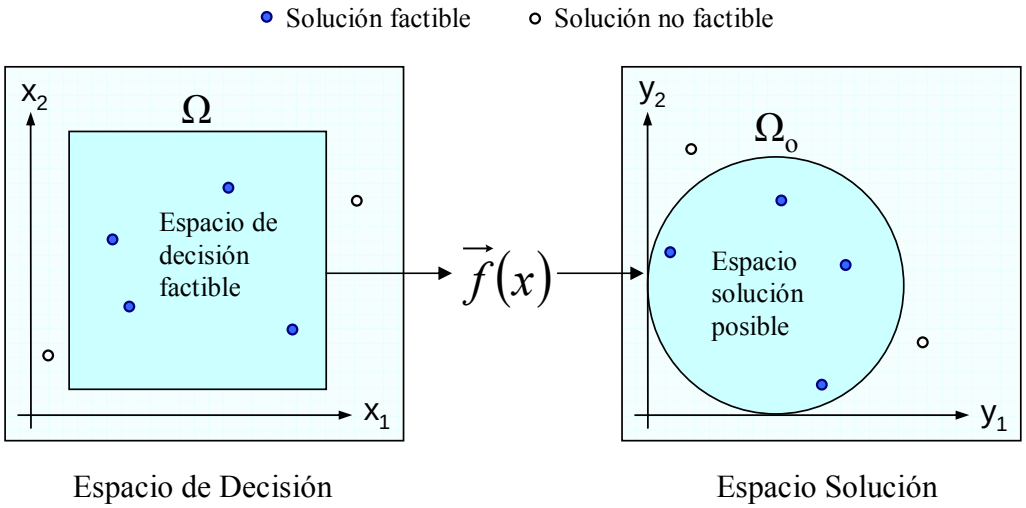


Figura 2.1. Ilustración de un MOP con dos objetivos y dos variables de decisión.

2.3 Optimalidad Pareto

Como fuera mencionado anteriormente, al resolver un MOP con al menos dos objetivos en conflicto encontramos que generalmente no existe una única solución óptima, sino que existen varias soluciones que representan las mejores combinaciones entre las evaluaciones de los objetivos, en donde ninguna solución puede ser considerada mejor que las demás y que por lo tanto todas son óptimas para el problema [Coello99, Veldhuizen99, Zitzler00].

Mediante los conceptos de *Optimalidad Pareto* (introducidos por el economista Vilfredo Pareto en 1896 [Toscano01]) es posible relacionar las diferentes soluciones para un MOP desde un punto de vista multiobjetivo. De manera que, estos conceptos permiten determinar si una o más soluciones son óptimas o no. Dichos conceptos son presentados a continuación [Toscano01, Lücken03, Zitzler00].

Definición 2.4 (dominancia Pareto): Sin pérdida de generalidad para el caso de un MOP de minimización, se dice que una solución $\vec{u} \in \Omega$ *domina* o *es mejor* que una solución $\vec{v} \in \Omega$, expresado como $\vec{u} \succ \vec{v}$, si y solo si:

$$f_i(\vec{u}) \leq f_i(\vec{v}) \ \ \forall\, i \in \{1,\ldots,M\} \ \text{ y además}$$
$$f_i(\vec{u}) < f_i(\vec{v}) \ \text{ para al menos un valor de } i. \tag{2.4}$$

Menos restrictivamente, se dice que una solución $\vec{u}$ *domina débilmente* a otra solución $\vec{v}$, denotado por $\vec{u} \succeq \vec{v}$, si y solo si:

$$f_i(\vec{u}) \leq f_i(\vec{v}) \ \ \forall\, i \in \{1,\ldots,M\}. \tag{2.5}$$

Por otra parte, si $\vec{u}$ no domina a $\vec{v}$ y tampoco $\vec{v}$ domina a $\vec{u}$, se dice que son soluciones *no comparables*, *no dominadas entre si* o *igualmente factibles*, lo que se denota como $\vec{u} \sim \vec{v}$.

Definición 2.5 (óptimo Pareto): Un vector de decisión $\vec{u} \in \Omega$ se dice que es un *óptimo Pareto* si y solo si no existe otro vector de decisión $\vec{v} \in \Omega$ tal que $\vec{v} \succ \vec{u}$. Estas soluciones no dominadas son llamadas *soluciones óptimas Pareto*.

Definición 2.6 (conjunto Pareto): El conjunto de todos los vectores de decisión óptimos Pareto se denomina *conjunto Pareto* y se denota por:

$$CP = \left\{ \vec{x} \mid \vec{x} \in \Omega \wedge \nexists \vec{x}' \in \Omega : \vec{x}' \succ \vec{x} \right\}.$$

Definición 2.7 (frente Pareto): La imagen de CP a través de $\vec{f}(\vec{x})$ en el espacio objetivo es el conjunto de todos los vectores óptimos de solución que se conoce como *frente Pareto* y se denota por:

$$FP = \left\{ \vec{y} \mid \vec{y} \in \Omega_0 \wedge \vec{y} = \vec{f}(\vec{x}) \ \forall \vec{x} \in CP \right\}.$$

En la figura 2.2 se muestra una ilustración de los conceptos mencionados para el caso de un MOP de minimización de dos objetivos y dos variables de decisión.

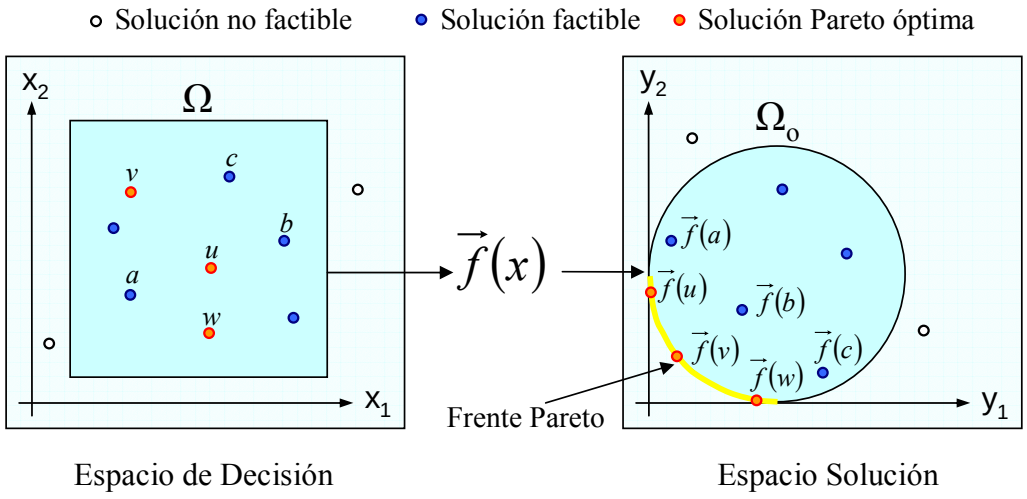


Figura 2.2. Ilustración de los conceptos de Optimalidad Pareto en un MOP de minimización.

Las soluciones $\vec{u}$, $\vec{v}$ y $\vec{w}$ son óptimos Pareto, mientras que las soluciones $\vec{a}$, $\vec{b}$, y $\vec{c}$ son soluciones factibles que no se dominan entre si, pero que son dominadas por cada una de las primeras soluciones respectivamente. El segmento resaltado del borde del conjunto Ω_o corresponde al frente Pareto del problema, en un contexto de minimización.

2.4 Resolución de Problemas de Optimización Multiobjetivos

La resolución de MOPs se puede realizar en dos fases que pueden ser implementadas tanto de manera separada como de manera conjunta. Estas fases son [Veldhuizen99, Lücken03]:

- La fase de *Optimización* o *Búsqueda de Soluciones*: que consiste en la búsqueda de una o más soluciones óptimas para el MOP, y
- La fase de *Toma de Decisiones*: que se refiere a la selección específica de una o más de las soluciones encontradas para ser aplicadas al caso concreto.

De esta forma, desde el punto de vista en que se dan o no estas fases es posible clasificar a los métodos de resolución de MOPs de la siguiente manera [Veldhuizen99, Toscano01, Lücken03]:

- *Métodos sin preferencia*: Esta clase incluye a los métodos en los que no se necesita realizar un proceso de Toma de Decisiones y se puede satisfacer la necesidad de resolución del MOP con cualquier solución óptima en el sentido Pareto (cualquier solución no dominada encontrada).
- *Métodos a priori*: En esta clase de métodos la especificación de las preferencias entre los objetivos es conocida de antemano, por lo cual la fase de Toma de Decisiones está dada antes que la fase de Optimización. Así es posible, (i) realizar una ponderación escalar de las evaluaciones de los objetivos de manera a utilizar una función de costo que los combine linealmente transformando el MOP en un problema de optimización mono-objetivo, o (ii) realizar la búsqueda de soluciones optimizando independientemente cada objetivo siguiendo cierto orden establecido en función a las preferencias entre los objetivos.
- *Métodos interactivos o progresivos*: En los métodos de esta clase las fases de Toma de Decisiones y de Optimización son realizadas de manera intercalada. De esta forma, se procede a la búsqueda de una o más soluciones óptimas, considerando entre uno o todos los objetivos, para luego realizar una selección de las soluciones encontradas en base a cierta información de preferencias sobre los objetivos que se ingresan en ese momento y de no encontrase una o más soluciones deseadas, se procede a la búsqueda de nuevas soluciones en base a las preferencias ingresadas y/o a las soluciones encontradas.
- *Métodos a posteriori*: En los métodos de esta clase la fase de Optimización preceda a la fase de Toma de Decisiones. De esta forma, los métodos de esta clase se ocupan de la generación de un grupo con la mayor cantidad de soluciones no dominadas entre si,

denominado *aproximación al frente Pareto del problema*, el cual luego es utilizado como entrada del proceso de Toma de Decisiones.

Los métodos a priori tienen la ventaja de que con ellos se puede tratar a los MOPs como problemas de optimización de un solo objetivo, pero requieren de un conocimiento profundo de los espacios de decisión y de solución de cada MOP a resolver [Lücken03].

Los métodos de resolución a posteriori permiten tener una visión completa del dominio del problema mediante la búsqueda de una aproximación al frente Pareto real del MOP, pero al no tener en cuenta las preferencias de la Toma de Decisiones necesitan explorar gran parte del espacio de búsqueda, con lo cual aumenta el tiempo y esfuerzo para la resolución de MOPs [Lücken03].

Por otra parte, los métodos interactivos o progresivos tienen la ventaja de permitir realizar búsqueda de soluciones atendiendo a las preferencias de la Toma de Decisiones [Lücken03]. En contrapartida, las implementaciones de esta clase de métodos son por lo general específicos al MOP que se intenta resolver, por lo que su adaptación para resolver otros MOPs puede resultar difícil de realizar.

2.5 Métodos Tradicionales de Resolución de Problemas Multiobjetivos

Los métodos tradicionales de resolución de MOPs se basan en tratar a estos problemas como problemas de optimización mono-objetivos, por lo cual son clasificados como *métodos a priori.*

Las desventajas de estos métodos radican en que es necesario un conocimiento previo acerca de los espacios de búsqueda y de solución del MOP a resolver y que para encontrar un conjunto de n soluciones no dominadas se requiere como mínimo de n instancias de ejecución del método utilizado [Lücken03].

A continuación se presentan algunos métodos tradicionales de resolución de MOPs.

2.5.1 Agregación Ponderada

El método de Agregación Ponderada [Lücken03] consiste en la combinación lineal de las evaluaciones de las funciones objetivo utilizando para ello diferentes coeficientes de ponderación, cuyos valores son determinados a partir de un conocimiento a priori sobre el espacio de decisión del MOP. Así, en este método la función objetivo vectorial del MOP se transforma en:

$$F(\vec{x}) = w_1 \times f_1(\vec{x}) + w_2 \times f_2(\vec{x}) + ... + w_M \times f_M(\vec{x}) \tag{2.6}$$

sujeto a:

$$\vec{g}(\vec{x}) = [g_1(\vec{x}), g_2(\vec{x}), \ldots, g_J(\vec{x})] \geq 0, \tag{2.7}$$

$$\vec{h}(\vec{x}) = [h_1(\vec{x}), h_2(\vec{x}), \ldots, h_K(\vec{x})] = 0, \tag{2.8}$$

donde los $0 \leq w_i \leq 1$ son las ponderaciones de cada objetivo y la suma de todos ellos debe ser igual a 1.

El método de la agregación ponderada se vuelve a subdividir de nuevo en tres enfoques [Parsopoulos02]:

- *Agregación Ponderada Convencional:* En la cual los pesos asignados a los objetivos se mantienen constantes durante el proceso de búsqueda.
- *Agregación Ponderada Dinámica:* En la cual los pesos asignados a los objetivos son modificados gradualmente durante el proceso de búsqueda. A continuación se muestra un ejemplo de la variación de los coeficientes de ponderación en un MOP de dos objetivos:

  $$w_1 = |\, sin\,(2\pi t / F)|, \; w_2 = 1 - w_1 \tag{2.9}$$

  donde t es el número de iteración y F es la frecuencia de cambio de los pesos.
- *Agregación Ponderada Bang-Bang:* En la cual los pesos asignados a cada objetivo cambian abruptamente durante el proceso de búsqueda. A continuación se muestra un ejemplo de la variación de los coeficientes de ponderación para un MOP de dos objetivos:

  $$w_1 = sigm\,(sin\,(2 \times \pi \times t / F)), \; w_2 = 1 - w_1, \tag{2.10}$$

  $$sigm\,(a) = \frac{1}{1 + e^{-a}}\,. \tag{2.11}$$

2.5.2 Programación de Metas

El método de Programación de Metas [Lücken03] consiste en asignar los valores que se quieren obtener en cada objetivo como metas adicionales al problema. De esta forma, se trata de minimizar las desviaciones absolutas de las evaluaciones de los objetivos de acuerdo con sus respectivas metas. La forma más simple de este método se plantea de la siguiente manera:

$$Optimizar \quad F(\vec{x}) = \sum_{i=1}^{M} \left| \frac{f_i(\vec{x}) - T_i}{T_i} \right|, \tag{2.12}$$

donde $\vec{x}$ es un vector de decisión que pertenece al espacio de decisión factible y las T_i son las metas establecidas para cada objetivo.

Esta técnica puede resultar muy eficiente si las metas impuestas a los objetivos se encuentran dentro del espacio solución posible y además se conoce la prioridad en que las metas deben ser cumplidas [Lücken03].

2.5.3 Ordenamiento Lexicográfico

En el método de Ordenamiento Lexicográfico [Lücken03] cada función objetivo se optimiza independientemente a las demás, siguiendo un orden secuencial derivado de la importancia relativa de cada objetivo sobre los demás según las preferencias de la Toma de Decisiones.

Suponiendo que los subíndices que identifican a las funciones objetivos también indican su orden de optimización, un MOP se resuelve a través de este método como:

$$Optimizar \;\; f_1(\vec{x}) \tag{2.13}$$

sujeto a:

$$\vec{g}(\vec{x}) \;=\; [g_1(\vec{x}),\, g_2(\vec{x}),\ldots,\, g_J(\vec{x})] \;\geq\; 0, \tag{2.14}$$

$$\vec{h}(\vec{x}) = [h_1(\vec{x}),\, h_2(\vec{x}),\ldots,\, h_K(\vec{x})] \;=\; 0, \tag{2.15}$$

obteniéndose la solución $\vec{x}_1^*$ que optimiza la función $f_1(\vec{x})$. Posteriormente, se sigue con la resolución del MOP como:

$$Optimizar \;\; f_2(\vec{x}) \tag{2.16}$$

sujeto a:

$$\vec{g}(\vec{x}) \;=\; [g_1(\vec{x}),\, g_2(\vec{x}),\ldots,\, g_J(\vec{x})] \;\geq\; 0, \tag{2.17}$$

$$\vec{h}(\vec{x}) = [h_1(\vec{x}),\, h_2(\vec{x}),\ldots,\, h_K(\vec{x})] \;=\; 0, \tag{2.18}$$

$$f_1(\vec{x}) = f_1(\vec{x}_1^*), \tag{2.19}$$

consiguiéndose la solución $\vec{x}_2^*$ que optimiza la función $f_2(\vec{x})$. Así, la resolución del MOP concluye en la optimización de la última función objetivo:

$$Optimizar \;\; f_M(\vec{x}) \tag{2.20}$$

sujeto a:

$$\vec{g}(\vec{x}) \;=\; [g_1(\vec{x}),\, g_2(\vec{x}),\ldots,\, g_J(\vec{x})] \;\geq\; 0, \tag{2.21}$$

$$\vec{h}(\vec{x}) = [h_1(\vec{x}),\, h_2(\vec{x}),\ldots,\, h_K(\vec{x})] \;=\; 0, \tag{2.22}$$

$$f_1(\vec{x}) = f_1(\vec{x}_1^*),\; f_2(\vec{x}) = f_2(\vec{x}_2^*),\; \ldots,\; f_{M-1}(\vec{x}) = f_{M-1}(\vec{x}_{M-1}^*). \tag{2.23}$$

La solución $\vec{x}_M^*$ encontrada al final es considerada entonces como la solución deseada.

La utilización de este método es adecuada sólo cuando la importancia de cada objetivo sobre los demás es conocida de antemano [Lücken03].

2.5.4 Método de Restricciones ε

El Método de Restricciones ε (ypsilon) [Lücken03] se basa en la optimización de sólo una función objetivo, denotada como $f_P(\vec{x})$, considerando a las demás funciones objetivo en restricciones adicionales al problema en donde sus evaluaciones son acotadas a ciertos valores permisibles ε_i.

Así sin pérdida de generalidad, un MOP en un contexto de minimización se reformula a través de este método como:

$$Minimizar \quad f_P(\vec{x}) \tag{2.24}$$

sujeto a:

$$\vec{g}(\vec{x}) = [g_1(\vec{x}), g_2(\vec{x}),\ldots, g_J(\vec{x})] \geq 0, \tag{2.25}$$

$$\vec{h}(\vec{x}) = [h_1(\vec{x}), h_2(\vec{x}),\ldots, h_K(\vec{x})] = 0, \tag{2.26}$$

y a las restricciones adicionales sobre las funciones objetivo restantes:

$$f_i(\vec{x}) \leq \varepsilon_i \quad \forall i \in \{1,\ldots,M\} \wedge i \neq p\,. \tag{2.27}$$

Para encontrar una solución óptima a través de este método es necesario un conocimiento a priori de los espacios de búsqueda y de solución del MOP, de manera a escoger correctamente la función objetivo $f_P(\vec{x})$ y los valores apropiados para los valores permisibles ε_i. Además, para obtener diferentes soluciones óptimas el procedimiento debe repetirse asignando diferentes valores a las restricciones ε_i y al índice *p* [Lücken03].

2.6 Algoritmos Evolutivos

Los Algoritmos Evolutivos (EA – *Evolutionary Algorithms*) se basan en la simulación de un proceso de evolución Darwiniano sobre una población de individuos que representan soluciones potenciales al problema que se pretende resolver [Zitzler00, Lücken03]. Este proceso evolutivo está soportado a través de la aplicación iterativa de las operaciones de selección, cruzamiento, mutación y reproducción sobre los individuos de la población.

La operación de *selección* consiste en la identificación de los individuos que serán sometidos a las operaciones de cruzamiento y mutación. La operación de *cruzamiento* consiste en la generación de nuevos individuos a partir de la combinación de dos o más individuos de la población, donde los nuevos individuos se dice que poseen características que son similares o

heredadas de sus progenitores. La operación de *mutación* consiste en la modificación de la información contenida en el individuo de manera aleatoria, la cual suele realizarse de acuerdo a cierta probabilidad. Por último, la operación de *reproducción* consiste en la identificación de los individuos más aptos en la población actual que conformarán la población para la siguiente generación.

A continuación se muestra el pseudocódigo genérico de un EA:

```
1.  t := 0;
2.  Inicializar P(0);
3.  Evaluar P(0);
4.  DO
5.     P'(t) := Cruzar(Seleccionar(P(t));
6.     Mutar(P'(t));
7.     Evaluar(P'(t));
8.     P(t+1) := Reproducir(P(t) U P'(t));
9.     t := t+1;
10. WHILE(condición_de_parada() == FALSE)
11. Retornar mejor solución;
```

Figura 2.3. Pseudocódigo genérico de un Algoritmo Evolutivo.

Los EAs comienzan con la creación, inicialización y evaluación de una población inicial de individuos generados al azar, la cual será refinada mediante la aplicación iterativa de las operaciones de cruzamiento, mutación, selección y reproducción.

La operación de *evaluación* consiste en la obtención de un *valor de adaptabilidad* (*fitness*) para cada individuo, el cual indica que tan buena es la solución representada por el individuo para el problema que se pretende resolver.

Así, en cada generación t se genera una población de individuos $P'(t)$ mediante la aplicación de la operación de cruzamiento sobre un conjunto de individuos seleccionados a partir de su valor de adaptabilidad de la población $P(t)$. Luego, los individuos de esta población $P'(t)$ son sometidos a la operación de mutación. Por último, mediante el proceso de reproducción se determinan los individuos de $P(t)$ y $P'(t)$ que conformarán la población para la siguiente generación $P(t+1)$.

La función *condición_de_parada*() devuelve un valor booleano que indica la terminación de las iteraciones del EA, lo cual se da una vez que se haya encontrado una solución deseada, haya transcurrido un cierto número de generaciones o se alcance un cierto tiempo de ejecución.

Cabe hacer notar que la manera en que se aplican e implementan los operadores de selección, cruzamiento, mutación y reproducción diferencia a las distintas técnicas basadas en la teoría de EA, la cual se divide en los paradigmas descritos a continuación [Lücken03, Toscano01].

2.6.1 Algoritmos Genéticos

Los Algoritmos Genéticos (GA – *Genetic Algorithm*), introducidos por Holland [Holland92], constituyen una de las técnicas evolutivas de mayor utilización en la actualidad. Esta técnica trata la aplicación de mecanismos computacionales inspirados en los procesos de evolución biológica observados a nivel genético, tales como selección, mutación y recombinación genética (cruzamiento de individuos), sobre una población de soluciones potenciales al problema que se pretende resolver.

Para poder aplicar los GAs en la resolución de problemas se requiere de la definición de los siguientes componentes básicos [Toscano01]:

- Una *forma de representación* de las soluciones potenciales al problema, denominadas *individuos*. Típicamente mediante una cadena de símbolos con cierto tipo de codificación (binaria, real, entera, alfanumérica) a la cual se denomina *cromosoma*. A cada posición dentro de los cromosomas se denomina *gen*, el cual debe ser capaz de almacenar un elemento de cierto conjunto de valores denominados *alelos*.
- Una forma de crear una *población inicial* de posibles soluciones al problema.
- Una *función de evaluación* que permita calcular un valor de adaptabilidad de los individuos de la población.
- Las definiciones de las formas de realización y aplicación de las o*peraciones genéticas* de selección, recombinación, mutación y reproducción.
- El establecimiento de los valores para los parámetros del GA tales como tamaño de la población, probabilidad de cruce, probabilidad de mutación, número máximo de generaciones, etc.

El pseudocódigo genérico para los GAs es el mismo que el presentado en la figura 2.3.

2.6.1 Estrategias Evolutivas

La Estrategias Evolutivas (ES – *Evolution Strategies*) [Toscano01, Beyer02, Lara03] se basan en la evolución mediante la aplicación de un proceso de mutación sobre un individuo, el cual posiblemente fue creado a partir de un proceso de recombinación genética.

La primera versión de este método, denominada Estrategia Evolutiva (1+1)–ES, no utiliza recombinación genética y el nuevo ser es obtenido a partir de la aplicación de un proceso de mutación sobre un individuo previamente seleccionado de la población.

El proceso de mutación se basa en la siguiente ecuación:

$$\vec{x}_n = \vec{x}_i^t + \vec{N}(0,\sigma)\,, \tag{2.28}$$

donde $\vec{x}_i^t$ es el individuo *i* seleccionado de la población en la generación *t* a partir del cual es creado el nuevo ser $\vec{x}_n$ y $\vec{N}(0,\sigma)$ es un vector de números gaussianos con valor medio cero y desviación estándar σ.

En esta primera versión un enfoque de selección extintiva es aplicado, donde el nuevo individuo reemplaza al padre si fuese mejor que este. Además, no solo las variables de decisión del problema son evolucionadas, sino que también el valor del parámetro σ, lo cual se realiza dependiendo del número de éxitos obtenidos en el proceso de mutación luego de cierta cantidad de generaciones, permitiéndose que tenga lugar un proceso de auto adaptación [Lara03].

En la figura 2.4 se muestra el pseudocódigo genérico para la Estrategia Evolutiva (1+1)–ES.

```
1.   t := 0;
2.   Generar la población inicial P(0);
3.   Evaluar P(0);
4.   DO
5.      x := Seleccionar un individuo de P(t);
6.      xn := x + N(0, σ);  /* mutación de x */
7.      Evaluar xn ;
8.      Reemplazar x con xn si este es mejor;
9.      Ajustar σ;
10.     t := t+1;
11.  WHILE(condición_de_parada() == FALSE)
12.  Retornar mejor solución;
```

Figura 2.4. Pseudocódigo genérico de la Estrategia Evolutiva (1+1)–ES.

Posteriormente se propusieron varias variantes. La Estrategia Evolutiva (μ + 1)–ES, que consiste en la generación de un nuevo ser a partir de la recombinación de μ padres, el cual reemplaza al peor de ellos si este fuese mejor. La Estrategia Evolutiva (μ + λ)–ES que consiste en la generación de λ hijos a partir de la recombinación de μ padres, para luego elegir como siguiente generación a los μ mejores individuos de la unión de los μ padres y los λ hijos. Por último, la Estrategia Evolutiva (μ, λ)–ES que al igual que la anterior, consiste en la generación de λ hijos a partir de μ padres, eligiendo a los μ mejores hijos como la población para la siguiente generación.

Los pseudocódigos genéricos para las distintas variantes de la ES son mostrados en las figuras 2.5, 2.6 y 2.7.

```
1.    t := 0;
2.    Generar la población inicial P(0) de μ individuos;
3.    Evaluar P(0) ;
4.    DO
5.       Recombinar los μ individuos para formar xn;
6.       Mutar xn;
7.       Evaluar xn;
8.       Reemplazar al peor padre si xn este fuese mejor que este;
9.       Ajustar σ;
10.      t := t + 1;
11.   WHILE(condición_de_parada() == FALSE)
12.   Retornar mejor solución;
```

Figura 2.5. Pseudocódigo genérico de la Estrategia Evolutiva (μ+1)–ES.

```
1.    t := 0;
2.    Generar la población inicial P(0) de μ individuos;
3.    Evaluar P(0);
4.    DO
5.       Recombinar los μ individuos para formar λ hijos;
6.       Mutar los λ hijos;
7.       Evaluar los λ hijos;
8.       P(t+1) := mejores μ individuos de entre padres e hijos;
9.       Ajustar σ;
10.      t := t+1;
11.   WHILE(condición_de_parada() == FALSE)
12.   Retornar mejor solución;
```

Figura 2.6. Pseudocódigo genérico de la Estrategia Evolutiva (μ+λ)–ES.

```
1.    t := 0;
2.    Generar la población inicial P(0) de μ individuos;
3.    Evaluar P(0);
4.    DO
5.       Recombinar los μ individuos para formar λ hijos;
6.       Mutar los λ hijos;
7.       Evaluar los λ hijos;
8.       P(t+1) := mejores μ hijos;
9.       Ajustar σ;
10.      t := t+1;
11.   WHILE(condición_de_parada() == FALSE)
12.   Retornar mejor solución;
```

Figura 2.7. Pseudocódigo genérico de la Estrategia Evolutiva (μ, λ)–ES.

2.6.2 Programación Evolutiva

La técnica de Programación Evolutiva (EP – *Evolutionary Programming*), introducida por Fogel [Fogel64], se inspira en la evolución de individuos de distintas especies en donde el proceso evolutivo se soporta sólo a través del operador de mutación como en los métodos ES.

En los métodos EP no existe un modelo estricto de representación de los individuos de la población, sino que estos se pueden representar computacionalmente de diferentes formas, por ejemplo como matrices, listas, árboles, programas, etc. Las únicas restricciones impuestas son que los individuos puedan ser sometidos a un proceso de mutación basado en simples cambios sobre su representación de acuerdo con cierta distribución estadística y que puedan ser evaluados de manera a determinar su valor de adaptabilidad con respecto al problema que se pretende resolver.

Debido a que la EP es una abstracción del proceso evolutivo en el cual se dice que los individuos de la población pertenecen a diferentes especies, la recombinación entre los individuos de la población no ocurre en este método.

Por otra parte, la operación de reproducción consiste en una competencia de torneo entre dos o más individuos, de los cuales salen vencedores los individuos con mejor valor de adaptabilidad para el problema que se pretende resolver.

El pseudocódigo genérico del método EP es presentado en la figura a continuación.

```
1.  t := 0;
2.  Generar la población inicial P(0) de Individuos;
3.  Evaluar P(0);
4.  DO
5.     P'(t) := Mutar los individuos de P(t);
6.     Evaluar P'(t);
7.     P(t+1) := Reproducir individuos más aptos de P(t) y P'(t);
8.     t := t+1;
9.  WHILE(condición_de_parada() == FALSE)
10. Retornar mejor solución;
```

Figura 2.8. Pseudocódigo genérico del método de Programación Evolutiva.

2.6.3 Programación Genética

La Programación Genética (GP – *Genetic Programming*), propuesta inicial e independientemente por Koza [Koza89] y Cramer [Cramer85], consiste en la evolución de programas computacionales mediante un proceso evolutivo.

En esta técnica, los individuos son programas que intentan resolver un cierto problema y se representan mediante estructuras de árboles que son sometidas a los procesos de cruzamiento y mutación.

En la figura que sigue se ilustra el pseudocódigo genérico para la técnica de GP.

```
1.  t:= 0;
2.  Generar la población inicial P(0) de programas;
3.  DO
4.     Evaluar cada individuo de P(t);
5.     P'(t) := Seleccionar P(t) individuos en base a su evaluación;
6.     Cruzar y mutar individuos de P'(t);
7.     P(t+1) := Reproducir individuos más aptos de P(t) y P'(t);
8.     t := t+1;
9.  WHILE(condición_de_parada() == FALSE)
10. Retornar mejor solución;
```

Figura 2.9. Pseudocódigo genérico de la técnica de Programación Genética.

Todos los programas en la población inicial han de ser sintácticamente correctos y los operadores genéticos también han de producir programas sintácticamente correctos.

El operador de cruzamiento combina dos o más individuos intercambiando entre ellos subárboles de sus estructuras elegidos al azar, para generar uno o más nuevos individuos.

El operador de mutación actúa sobre un solo individuo y por lo general consiste en la generación o eliminación de un subárbol a partir de un nodo elegido al azar, o en el intercambio de subárboles dentro de la estructura del individuo.

En la figura 2.10 se ilustran ejemplos de las operaciones de cruzamiento y mutación de programas.

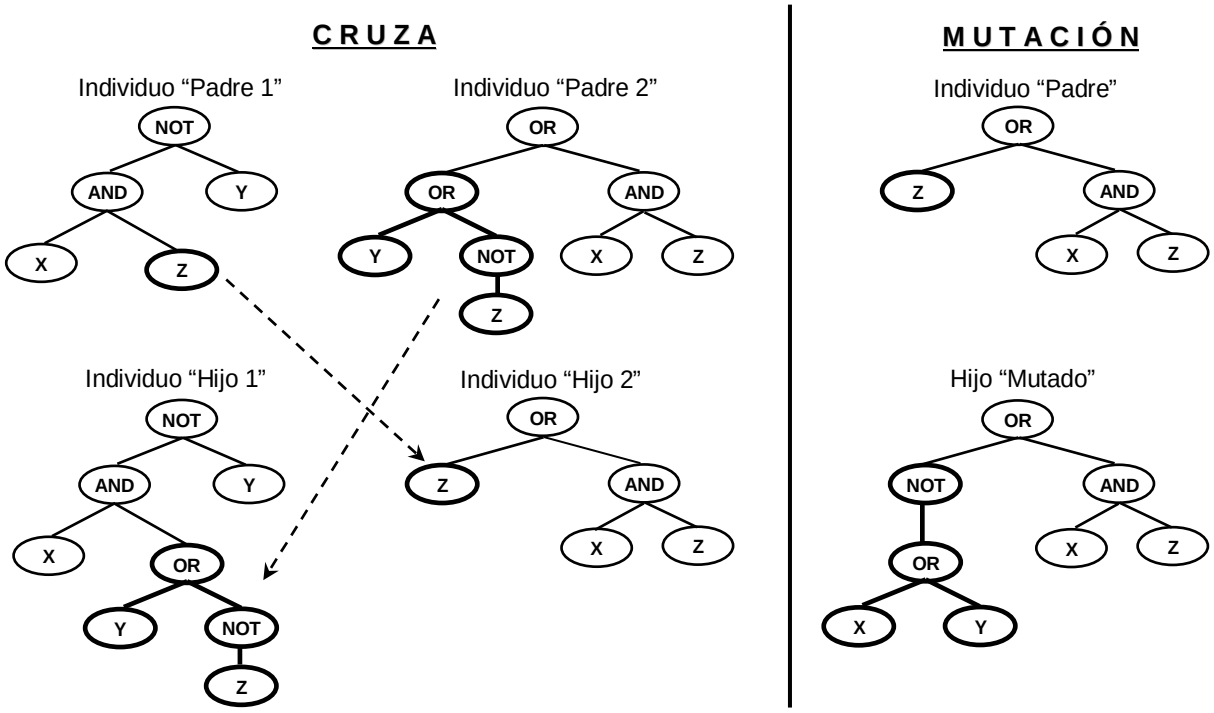


Figura 2.10. Ilustración de las operaciones de cruzamiento y mutación en la Programación Genética.

2.7 Métodos basados en Algoritmos Evolutivos para Problemas Multiobjetivos

Debido a la dificultad para obtener soluciones óptimas para ciertos MOPs a través de métodos tradicionales, sin necesitar más de una instancia de ejecución y sin poseer un conocimiento previo sobre los espacios de decisión y de solución del MOP [Lücken03], surge la necesidad de métodos alternativos de resolución de MOPs que cubran estas falencias.

Los EA para MOPs aparecen entonces como un tipo de métodos que logran cubrir estas debilidades de los métodos clásicos. Estos métodos caen dentro de la clasificación de *métodos a posteriori*, cuyo resultado consiste como mínimo en una *aproximación al frente Pareto real* del MOP que se pretende resolver.

A continuación se presentan algunos de métodos más utilizados basados en la teoría de EA para la resolución de MOPs.

2.7.1 Nondominated Sorting Genetic Algorithm

El método *Nondominated Sorting Genetic Algorithm* (NSGA), propuesto por Srinivas et al. [Srinivas94], difiere del algoritmo genético simple solo en la manera en como se asigna el valor de adaptabilidad a los individuos de la población para realizar el proceso de selección.

En cada generación los individuos de la población se clasifican en varios frentes. Primero se identifican todos los individuos no dominados de la población, los cuales constituyen el primer frente y se les asigna un mismo valor de adaptabilidad inicial llamado *dummy fitness*. Luego, para cada individuo del primer frente es calculado un segundo valor de adaptabilidad denominado *shared fitness*, el cual se halla dividiendo el valor del *dummy fitness* por un valor proporcional a la cantidad de individuos que se encuentran a su alrededor. Ahora los individuos del primer frente son ignorados y se procede a determinar el segundo frente siguiendo el mismo procedimiento, pero asignando a los individuos un valor de *dummy fitness* ligeramente menor que el menor valor de *shared fitness* encontrado en el primer frente. Este proceso se repite hasta que todos los individuos hayan sidos clasificados.

La reducción del valor de adaptabilidad inicialmente asignado (*dummy fitness*) constituye una estrategia denominada *fitness sharing*, que se basa en la idea de la competencia entre los individuos por los recursos disponibles en la zona en que se encuentran [Srinivas94]. El objetivo de usar una estrategia de *fitness sharing* es evitar la aglomeración de los individuos intentando lograr una mayor distribución de estos en el espacio solución. Así, el valor de *shared fitness* s_i para cada individuo i de la población, considerando una cierta distancia de separación σ_{share} entre ellos, es calculado conforme a:

$$s_i = \frac{d_i}{\sum_{j=1}^{n} sh\left(d\left(i,j\right)\right)}, \tag{2.29}$$

donde d_i es el valor del *dummy fitness* del individuo i, n es el número de individuos de la población, $d(i, j)$ es una función que calcula la distancia euclidiana entre los vectores de evaluación de los individuos i y j, mientras que la función $sh()$ se define como:

$$sh\left(d\left(i,j\right)\right) = \begin{Bmatrix} 1-\left(\dfrac{d\left(i,j\right)}{\sigma_{share}}\right)^2 & \text{si } d\left(i,j\right) < \sigma_{share} \\ 0 & \text{en caso contrario} \end{Bmatrix}. \tag{2.30}$$

Luego de la clasificación, los individuos son seleccionados para la aplicación de la operación de reproducción considerando sus valores de *dummy fitness* y las operaciones de cruza-

miento y mutación a través de sus valores de *shared fitness*, de manera a determinar la población para la siguiente generación.

En la figura 2.11 se muestra el pseudocódigo genérico para el método NSGA.

```
1.    t := 0;
2.    Generar la población inicial P(0);
3.    DO
4.       f := 1
5.       WHILE (P(t) no clasificada) DO
6.          Identificar individuos para el frente f;
7.          Asignar el valor de dummy fitness a los individuos del frente f;
8.          Hallar el valor de shared fitness para cada individuo del frente f;
9.          f := f+1;
10.      END_WHILE
11.      P(t+1) := Reproducir P(t) según dummy fitness,
                   Cruzar y Mutar P(t) según shared fitness;
12.      t := t+1;
13.   WHILE(condición_de_parada() == FALSE)
14.   Retornar soluciones no dominadas en P(t);
```

Figura 2.11. Pseudocódigo genérico del método NSGA.

2.7.2 Niched Pareto Genetic Algorithm

El *Niched Pareto Genetic Algorithm* (NPGA), propuesto por Horn et al. [Horn93, Horn94], difiere del algoritmo genético simple solo en el proceso de selección de los individuos.

En este método se propone la utilización de un procedimiento de selección de individuos denominado *torneo por dominancia Pareto*, que utiliza una estrategia de *fitness sharing* [Horn94] para romper empates. Este procedimiento de selección tiene por objetivo identificar un subconjunto de individuos de la población genética, denominado *conjunto de apareamiento* (*mating pool*), a ser utilizado para la aplicación posterior de los operadores de cruzamiento, mutación y reproducción, cuyos resultados constituyen la población para la siguiente generación.

En el procedimiento de *torneo por dominancia Pareto* se seleccionan de la población genética de forma aleatoria dos individuos candidatos i y j, y un conjunto de comparación P_{dom}. Cada candidato se compara contra cada elemento del conjunto comparación por medio de la relación de dominancia. Si un candidato es dominado por el conjunto comparación y el otro no, se selecciona como ganador al candidato no dominado.

Cuando ninguno o ambos candidatos son dominados por el conjunto de comparación P_{dom}, entonces se utiliza la estrategia de *fitness sharing* para romper el empate y elegir un ganador. Esta estrategia consiste en calcular el número de individuos en el conjunto de apareamiento que se encuentran alrededor de cada solución candidata considerando cierta distancia euclidiana de separación σ_{share} [Horn94]. Como se intenta mantener la diversidad en la población, el candidato que tenga el menor número de individuos a su alrededor es seleccionado como el ganador. Si el empate persiste, cualquiera de ellos es elegido como ganador.

Dado que los individuos candidatos son comparados mediante la relación de dominancia contra un subconjunto de individuos de la población genética, el suceso del algoritmo depende del tamaño de este subconjunto. Un conjunto de comparación pequeño puede provocar que muy pocas soluciones no dominadas sean encontradas, mientras que la utilización de un conjunto de comparación grande puede resultar en un caso de convergencia prematura [Horn94].

El pseudocódigo genérico para el método NPGA se muestra en la figura a continuación.

```
1.   t:= 0;
2.   Generar la población inicial P(0);
3.   DO
4.      M(t) := ∅;  /* mating pool */
5.      WHILE(mating_pool_seleccionado() == FALSE) DO
6.         Seleccionar dos individuos candidatos P[i] y P[j];
7.         Seleccionar el conjunto de comparación Pdom;
8.         IF(Pdom NO_DOMINA_A P[i] AND Pdom DOMINA_A P[j])
9.            M(t) := M(t) ∪ P[i];
10.        ELSE IF(Pdom DOMINA_A P[i] AND Pdom NO_DOMINA_A P[j])
11.           M(t) := M(t) ∪ P[j];
12.        ELSE IF(sharing(P[i]) < sharing(P[i]))
13.           M(t) := M(t) ∪ P[i];
14.        ELSE IF(sharing(P[j]) < sharing(P[i]))
15.           M(t) := M(t) ∪ P[j];
16.        ELSE
17.           Elegir al alzar a un candidato y agregarlo M(t);
18.        END_IF
19.     END_WHILE
20.     P(t+1) := reproducir, cruzar y mutar M(t);
21.     t := t+1;
22.  WHILE(condición_de_parada() == FALSE)
23.  Retornar soluciones no dominadas en P(t);
```

Figura 2.12. Pseudocódigo genérico del método NPGA.

2.7.3 Pareto Archived Evolutionary Strategy

El método *Pareto Archived Evolutionary Strategy* (PAES), propuesto por Knowles y Corne [Knwoles00], se basa en la Estrategia Evolutiva (1+1)–ES. Este método realiza una búsqueda local sometiendo a una solución a un proceso de mutación y también utiliza un repositorio que almacena las soluciones no dominadas encontradas durante su ejecución.

El método PAES comienza con la inicialización de un solo individuo el cual es evaluado e insertado en el repositorio. En cada iteración un individuo es seleccionado del repositorio, el cual es duplicado y su copia es sometida al proceso de mutación para producir un individuo candidato. Si el individuo candidato domina al individuo tomado del repositorio, este último es reemplazado por el primero. Si el individuo candidato resulta ser dominado es descartado. Por otra parte, si el individuo candidato es no comparable al individuo tomado del repositorio, entonces este es comparado contra los demás elementos del repositorio y si resulta no dominado por ellos se intenta insertarlo en el repositorio.

El repositorio de soluciones en el método PAES es de tamaño fijo y utiliza un esquema de actualización denominado *malla adaptativa*. En el esquema de *malla adaptativa* el espacio de solución explorado es representado por regiones dentro de la cuales son almacenadas las soluciones. La denominación de *adaptativa* proviene de la propiedad del repositorio de auto-adecuar los límites de estas regiones cuando se intenta insertar una solución que inicialmente no cae dentro de ninguna región.

La inserción de un nuevo elemento al repositorio es como sigue. Si el nuevo elemento domina a otros miembros del repositorio, entonces estos elementos dominados son removidos y se inserta el nuevo elemento. Si el nuevo elemento no es dominado y no domina a ningún miembro del repositorio cayendo dentro de una región muy poblada cuando el repositorio está lleno es descartado, caso contrario es insertado. Por el contrario, si el nuevo individuo cae dentro una región poco poblada es insertado y si el repositorio estuviese lleno se descarta una solución de la región más poblada.

En el pseudocódigo genérico del método PAES se muestra en la figura que sigue.

```
1.  R := ∅;
2.  generar una solución e insertarla en el repositorio R;
3.  DO
4.     Seleccionar una solución X de R;
5.     X' := mutar X;
6.     IF(X' DOMINA_A X OR (X NO_DOMINA_A X' AND R NO_DOMINA_A X'))
7.        /* insertar X' en R */
8.        IF(X' domina a elementos de R)
9.           Eliminar elementos dominados de R;
10.          Agregar X' a R;
11.       ELSE IF(X' cae en una región no muy poblada OR R no está lleno)
12.          IF(R está lleno)
13.             Eliminar una solución de la región más poblada de R;
14.          END_IF
15.          Agregar X' a R;
16.       END_IF
17.    END_IF
18. WHILE(condición_de_parada() == FALSE)
19. retornar soluciones no dominadas en R;
```

Figura 2.13. Pseudocódigo genérico del método PAES.

2.7.4 Strength Pareto Evolutionary Algorithm

El *Strength Pareto Evolutionary Algorithm* (SPEA), propuesto por Zitzler et al. [Zitzler98], es un método basado en el algoritmo genético simple que posee varias características. (i) Almacena las soluciones no dominadas encontradas en una población externa, las cuales junto con los individuos de la población genética participan del proceso de selección. (ii) Asigna un valor de adaptabilidad escalar a los individuos basado en el concepto de dominancia Pareto, el cual también ayuda a preservar la diversidad en la población genética. (iii) Mantiene constan-

te el número soluciones no dominadas almacenadas en la población externa sin destruir las características del frente aproximado.

En cada generación del método SPEA, los individuos de la población genética no dominados por los elementos de la población externa son copiados en esta última, eliminándose las soluciones que resulten dominadas. De esta forma, se preservan las buenas soluciones encontradas durante el proceso de búsqueda, a lo cual se denomina *elitismo*.

Debido a que la población externa interviene en los procesos de selección y de asignación del valor de adaptabilidad a los individuos de la población genética, esta debe limitarse a cierto tamaño de manera a que no disminuya la presión de selección provocando que la búsqueda de soluciones se ralentice [Zitzler98]. Así, cuando el tamaño de la población externa supera cierto limite, esta se reduce aplicando un procedimiento denominado *clustering*, que conserva las características del frente Pareto aproximado hasta el momento [Zitzler98] y cuyo pseudocódigo se ilustra en la figura 2.14.

```
1.  Recibir población externa de elementos no dominados E;
2.  max_elem := Recibir número máximo de elementos para E;
3.  /* se inicializa un conjunto de clusters */
4.  cluster_set := ∅;
5.  FOR cada elemento P en E DO
6.     cluster_set := cluster_set ⋃ {{P}};
7.  END_FOR
8.  WHILE |cluster_set| > max_elem DO
9.     min_dist = ∞;
10.    FOR cada par (X, Y) en cluster_set DO
11.       IF distancia_entre(X, Y) < min_dist THEN
12.          cluster_1 := X;
13.          cluster_2 := Y;
14.          min_dist  := distancia_entre(X, Y);
15.       END_IF
16.    END_FOR
17.    nuevo_cluster := cluster_1 ⋃ cluster_2;
18.    cluster_set   := cluster_set / {cluster_1, cluster_2};
19.    cluster_set   := cluster_set ⋃ nuevo_cluster;
20. END_WHILE
21. E := ∅;
22. FOR cada elemento c en cluster_set DO
23.    P := Obtener individuo central en C;
24.    E := E ⋃ P;
25. END_FOR
26. Retornar nuevo conjunto de elementos no dominados E;
```

Figura 2.14. Pseudocódigo del procedimiento de clustering.

El valor de adaptabilidad para cada individuo de la población externa se halla dividiendo el número de individuos a los que domina en la población genética por el tamaño de la población genética incrementado en uno. El valor de adaptabilidad de cada individuo de la población externa se obtiene calculando la inversa de su valor de *strength*.

El valor de adaptabilidad de cada individuo de la población genética también se obtiene a través de la determinación de su valor de *strength*, el cual se halla sumando los valores de

strength de los individuos de la población externa que lo dominan y luego incrementando el resultado de la suma en uno. Nuevamente, el valor de adaptabilidad de cada individuo es igual a la inversa de su valor de *strength*.

Mediante el procedimiento de cálculo del valor de adaptabilidad en la población externa, los individuos en dicha población que dominan a muchos individuos de la población genética reciben un bajo valor de adaptabilidad, con lo que se intenta evitar que a partir estos se generen nuevas soluciones que podrían resultar en la misma zona del espacio objetivo, no mejorándose así el frente Pareto aproximado hasta el momento [Zitzler98].

De la misma forma, los individuos de la población genética dominados por varios individuos de la población externa reciben un bajo valor de adaptabilidad, con lo que se intenta evitar que estos individuos den origen a nuevas soluciones que posiblemente resulten dominadas por las soluciones ya existentes [Zitzler98].

Luego de hallarse el valor de adaptabilidad para los individuos de la población externa y la población genética, se procede a identificar un conjunto de apareamiento (*mating pool*) a partir de la unión de estas poblaciones mediante una selección por torneo binario entre los individuos a través de sus valores de adaptabilidad, sobre los cuales posteriormente se aplican los operadores genéticos de cruzamiento y mutación.

Los individuos resultantes de la aplicación de las operaciones de cruzamiento y mutación sobre el *mating pool* constituyen la población para la siguiente generación.

El pseudocódigo genérico del método SPEA se muestra en la figura 2.15. Posteriormente este método fue mejorado por sus creadores [Zitzler02] al cual se denominó *Strength Pareto Evolutionary Algorithm 2* (SPEA2).

```
1.  t := 0;
2.  Generar la población inicial P(0);
3.  E := ∅; /* Población externa */
4.  DO
5.      Evaluar P(t);
6.      Actualizar población externa E;
7.      Reducir el tamaño E mediante procedimiento de clustering;
8.      M(t) := Seleccionar(P(t) ⋃ E); /* mating pool */
9.      P(t+1) := Cruzar y mutar M(t);
10.     t := t+1;
11. WHILE(condición_de_parada() == FALSE)
12. retornar soluciones no dominadas de E ⋃ P(t)
```

Figura 2.15. Pseudocódigo genérico del método SPEA.

2.7.5 Nondominated Sorting Genetic Algorithm II

El método NSGA [Srinivas94] posee varias falencias [Deb02]. (i) El procedimiento de clasificación de los individuos en varios frentes posee una alta complejidad computacional. (ii) El

método no presenta elitismo lo cual degrada su rendimiento al perder las buenas soluciones encontradas. (iii) Es necesaria la especificación a priori del valor para el parámetro σ_{share}.

Con la intención de solucionar estos problemas Deb et al. [Deb02] presentan una versión mejorada de este método denominada *Nondominated Sorting Genetic Algorithm II* (NSGA2).

El primer problema es enmendado en NSGA2 con la utilización de un nuevo procedimiento de clasificación de los individuos de la población genética en varios frentes, denominado *fast nondominated sorting procedure*, el cual se desarrolla como sigue. Primeramente se determina para cada individuo *i* de la población el número de individuos que lo dominan *ND*[*i*] y el conjunto *D*[*i*] de todos los individuos a los que domina. Todos los individuos *i* cuyo valor de *ND*[*i*] sea igual a cero conforman en primer frente en la población.

Para todos los elementos en los conjuntos *D*[*i*] de los individuos del primer frente se decrementa en uno su valor de *ND*[*i*] y aquellos individuos para los cuales su valor de *ND*[*i*] resulte igual a cero constituyen el segundo frente en la población. Ahora se sigue el mismo proceso con los elementos en los conjuntos *D*[*i*] de los individuos del segundo frente para determinar el tercer frente en la población. Este proceso se repite hasta clasificar todos los elementos de la población en sus respectivos frentes. El pseudocódigo para el *fast nondominated sorting procedure* se muestra en la figura 2.16.

```
1.  Recibir población de individuos P;
2.  FOR cada individuo p en P DO
3.     D[p] := ∅;
4.     FOR cada individuo q en P DO
5.        IF p DOMINA q THEN
6.           D[p] := D[p] ⋃ q;
7.        ELSE IF q DOMINA p THEN
8.           ND[p] := ND[p] + 1;
9.        END_IF
10.    END_FOR
11.    IF ND[p] := 0 THEN
12.       Frente[1] := Frente[1] ⋃ p;
13.    END_IF
14. END_FOR
15. f := 1
16. WHILE (Frente[f] != ∅)
17.    H := ∅;
18.    FOR cada p en Frente(f) DO
19.       FOR cada q en D[p] DO
20.           ND[q] := ND[q] - 1;
21.           IF(ND[q] = 0) THEN
22.              H := H ⋃ q;
23.           END_IF
24.       END_FOR
25.     f := f + 1;
26.     Frente[f] := H;
27. END_WHILE
28. Retornar los Frentes 1 a f-1;
```

Figura 2.16. Pseudocódigo para el fast nondominated sorting procedure.

El tercer problema de NSGA se soluciona en NSGA2 a través de un procedimiento que permite obtener una estimación de la densidad de soluciones alrededor de una solución en el espacio objetivo sin la necesidad de especificar el valor de algún parámetro para ello. Este procedimiento calcula para cada elemento *i* en cada frente un valor que indica la longitud lateral promedio del cuboide más grande que encierra a la solución *i* sin incluir a otra solución de su frente, este valor se denomina *crowding distance* y se denota como *distance*[*i*]. La figura 2.17 ilustra este caso en un espacio solución bidimensional. El pseudocódigo para el cálculo del *crowding distance* de cada individuo en un frente dado se muestra en la figura 2.18.

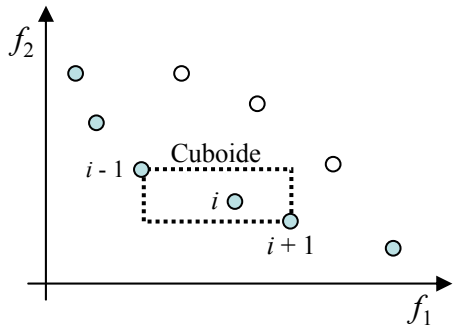


Figura 2.17. Cuboide para un individuo i en un espacio bidimensional.

```
1.  Recibir el frente F;
2.  L := Número de elementos en F;
3.  FOR i:= 1 TO L DO
4.     distance[i] := 0; /* crowding distance para el elemento i */
5.  END_FOR
6.  FOR m := 1 TO nro_de_objetivos DO
7.      Ordenar ascendentemente los elementos de F según el objetivo m;
8.      distance[1] := distance[L] := ∞; /* crowding para los elementos extremos */
9.      FOR i:= 2 TO L-1 DO
10.       distance[i] := distance[i] + evaluación[i+1][m] - evaluación[i-1][m];
11.     END_FOR
12. END_FOR
```

Figura 2.18. Pseudocódigo para el cálculo del crowding distance de cada individuo en un frente F.

Para evitar perder las buenas soluciones encontradas y mantener la población genética uniformemente distribuida a lo largo del frente Pareto aproximado, en NSGA2 se define un operador de comparación de individuos denominado *crowded comparison operator* ($\geq_n$), el cual guía al proceso de selección de los individuos definiendo un orden parcial (de mejor a peor) entre estos y se define como:

$$i \geq_n j \quad \text{si } (\textit{frente}[i] < \textit{frente}[j]) \text{ o } ((\textit{frente}[i] = \textit{frente}[j]) \text{ y } (\textit{distance}[i] > \textit{distance}[j])) ,$$

donde *frente*[*i*] indica el número del frente en el que se encuentra el individuo *i* y *distance*[*i*] indica el *crowding distance* del individuo *i.*

El pseudocódigo genérico para el método NSGA2 es presentado en la figura 2.19. Primeramente, una población inicial *P*(0) de *n* individuos es creada y clasificada en los varios fren-

tes mediante el *fast nondominated sorting procedure*. A cada individuo de la población se le asigna un valor de adaptabilidad igual a número de frente en que se encuentra y de esta forma se asume que lo deseado es la minimización del valor de adaptabilidad de los individuos.

A partir de este punto se inician las generaciones del algoritmo. Se realiza una selección por torneo binario de individuos de la población P(t) a través de su valor de adaptabilidad para la generar una población $P'(t)$ de n individuos mediante la aplicación de las operaciones de recombinación y mutación.

```
1.  t := 0;
2.  Generar la población inicial P(0) de n individuos;
3.  Clasificar P(0) mediante fast nondominated sorting procedure;
4.  DO
5.     P'(t) := seleccionar, cruzar y mutar P(t);
6.     R(t) := P(t) ⋃ P'(t);
7.     Clasificar R(t) mediante fast nondominated sorting procedure;
8.     P(t+1) := ∅;
9.     f := 1;
10.    WHILE(P(t+1) < n) DO /* n es el tamaño de la población */
11.       Calcular crowding distance para el frente(f) de R(t);
12.       Agregar individuos del frente(f) de R(t) a P(t+1);
13.    END_WHILE
14.    Ordenar P(t+1) mediante el crowded comparison operator;
15.    Reducir tamaño de P(t+1) a n elementos, eliminando los últimos elementos;
16.    t := t+1;
17. WHILE(condición_de_parada() == FALSE);
18. Retornar soluciones no dominadas de P(t);
```

Figura 2.19. Pseudocódigo genérico del método NSGA2.

La poblaciones $P(t)$ y $P'(t)$ son combinadas en una única población R, la cual es clasificada en varios frentes mediante el *fast nondominated sorting procedure*. Luego se halla el valor de *crowding distance* para los individuos de cada frente de R y se copian los individuos de cada frente en la población para la siguiente generación $P(t+1)$ hasta que se supere el número de n elementos. Nótese que los individuos de la población $P(t+1)$ ya tienen especificados sus valores de adaptabilidad debido a que ya fueron clasificados en sus correspondientes frentes.

Finalmente, los elementos en la población $P(t+1)$ son ordenados mediante el *crowded comparison operator* y se limita el tamaño de la población $P(t+1)$ a n elementos, tomando los n primeros (mejores) elementos.

2.7.6 Strength Pareto Evolutionary Algorithm 2

El método *Strength Pareto Evolutionary Algorithm 2* (SPEA2) es propuesto por Zitzler et al. [Zitzler02] como una versión mejorada del método SPEA [Zitzler98]. En particular, las principales diferencias del método SPEA2 con el método SPEA son [Zitzler02]: (i) un esquema mejorado de asignación del valor de adaptabilidad que tiene en cuenta para cada individuo el número de individuos que lo dominan y el número de individuos que domina, (ii) una técnica

para estimar la densidad de soluciones alrededor de cada individuo y (iii) un nuevo método de truncamiento de la población externa que evita la perdida de las soluciones en los limites del frente Pareto aproximado por esta población.

En el método SPEA dos individuos de la población genética dominados por los mismos elementos de la población externa reciben el mismo valor de adaptabilidad, independientemente de que uno ellos domine al otro. Como consecuencia, esto reduce la presión de selección haciendo que la efectividad del método SPEA se reduzca y en el caso de que la población externa conste de un solo individuo este método se comporta como un simple método de búsqueda aleatoria [Zitzler02].

Para evitar el problema anterior, en el método SPEA2 el valor de adaptabilidad para los individuos de la población externa $E(t)$ y de la población genética $P(t)$ se calcula como sigue. A cada individuo i en la unión de $E(t)$ y $P(t)$ se asigna un valor de *strength* $S(i)$ que se calcula como sigue:

$$S(i) = \left| \{ j \mid j \in P(t) \cup E(t) \wedge i \succ j \} \right|, \tag{2.31}$$

donde el valor de $S(i)$ indica el número de individuos que domina el individuo i. Luego en base a los valores de *strength* de los individuos, un valor denominado *raw fitness* $R(i)$ es hallado para cada elemento i en la unión de $E(t)$ y $P(t)$, el cual se calcula como sigue:

$$R(i) = \sum S(j) \;\; \forall \, j \in P(t) \cup E(t) \wedge j \succ i \,. \tag{2.32}$$

El valor $R(i)$ indica la fuerza que tienen sobre el individuo i los individuos de la población externa y la población genética que lo dominan. Debe notarse que este valor debe de ser minimizado.

Posteriormente se estima la densidad de las soluciones alrededor de cada individuo i en la unión de $P(t)$ y $E(t)$, calculándose la distancia euclidiana en el espacio objetivo de este individuo a los demás individuos j en la unión de $P(t)$ y $E(t)$, luego se ordenan estas distancias y se toma la distancia al k-ésimo elemento más cercano al individuo i, denotada como σ_i^k y donde k es igual al la raíz cuadrada de la cantidad de elementos en la unión de $P(t)$ y $E(t)$. De esta forma, la densidad $D(i)$ para cada individuo i se define como:

$$D(i) = \frac{1}{\sigma_i^k + 2}. \tag{2.33}$$

El valor de adaptabilidad $D(i)$ para cada individuo i en la unión de $P(t)$ y $E(t)$ se obtiene según:

$$F(i) = R(i) + D(i). \tag{2.34}$$

Esta forma de asignar el valor de adaptabilidad a los individuos evita que dos individuos dominados por las mismas soluciones reciban el mismo valor de adaptabilidad cuando uno de ellos domina al otro. También incluye información acerca de la densidad de las soluciones alrededor cada individuo en su valor de adaptabilidad, lo cual ayuda a guiar la búsqueda de soluciones más efectivamente [Zitzler02].

En el método SPEA el procedimiento de *clustering* ayuda a mantener constante el número de soluciones no dominadas en la población externa sin destruir las características del frente Pareto aproximado por estas soluciones. Pero este procedimiento puede hacer que se pierdan las soluciones en los límites del frente Pareto aproximado, las cuales deben ser mantenidas de manera a conseguir una buena distribución de soluciones a lo largo de este frente [Zitzler02].

En el método SPEA2, en cada generación t se determina una nueva población externa $E(t+1)$ de N' elementos, lo cual se realiza de la siguiente manera. Primero se copian todos los individuos no dominados de la unión de $P(t)$ y $E(t)$ en $E(t+1)$, cuyos valores de adaptabilidad son menores a 1. Si se completa el número N' de individuos requeridos en la población externa se finaliza el proceso. Si no se completa el número N' de elementos, se ordenan ascendentemente los elementos de la unión de $P(t)$ y $E(t)$ con valor de adaptabilidad mayor o igual a 1 y se insertan estos individuos en $E(t+1)$ según su orden hasta completar la cantidad de elementos requerida.

Por otra parte, si se sobrepasa el número N' de elementos en $E(t+1)$ se utiliza un proceso iterativo de remoción de elementos, en el cual en cada iteración se elimina el elemento i de $E(t+1)$ para el cual se cumpla la relación $i \leq_d j$, para todos los elementos j en $E(t+1)$. La relación $\leq_d$ se denomina *operador de truncamiento* y se define como:

$$\begin{aligned} i \leq_d j \quad \Leftrightarrow \quad & \forall\ 0 < k < \left|E(t+1)\right| : \sigma_i^k = \sigma_j^k \ \vee \\ & \exists\ 0 < k < \left|E(t+1)\right| : \sigma_i^k < \sigma_j^k \ \wedge \ \left(\forall\ 0 < l < k : \sigma_i^l = \sigma_j^l\right), \end{aligned} \tag{2.34}$$

donde σ_i^k denota la distancia entre el individuo i y el k-ésimo individuo más cercano a este individuo en $E(t+1)$. En otras palabras, el proceso de remoción de elementos elimina en cada iteración el elemento i en $E(t+1)$ más cercano a su k-ésimo vecino y en caso de empates entre varios individuos se considera sucesivamente la siguiente menor distancia para romper el empate [Zitzler02].

El pseudocódigo genérico para el método SPEA2 se ilustra en la figura 2.20. Primero se genera una población inicial $P(0)$ de N elementos y una población externa $E(0)$ vacía. Se asignan los valores de adaptabilidad a los elementos de la población genética y la población externa. Se genera una nueva población externa $E(t+1)$ de N' elementos y si se cumple la

condición de parada se retornan los elementos no dominados en la población externa $E(t+1)$. Sino se procede a hallar un *mating pool* $M(t)$ a partir de los elementos en la población externa $E(t+1)$ mediante la selección por torneo binario, para luego generar la población genética $P(t+1)$ mediante la aplicación de los operadores de cruzamiento y mutación.

```
1.  t := 0;
2.  Generar la población inicial P(0) de N individuos;
3.  E(0) := ∅;
4.  DO
5.     Asignar el valor de adaptabilidad a los individuos de P(t) y E(t);
6.     Generar E(t+1);
7.     IF(condición_de_parada() == FALSE) THEN
8.        BREAK; // romper ciclo infinito
9.     END_IF
10.    M(t) := selección por torneo binario de elementos de E(t+1);
11.    P(t+1) := cruzamiento y mutación de M(t);
12.    t := t+1
13. WHILE(TRUE); // ciclo infinito
14. Retornar soluciones no dominadas en E(t+1);
```

Figura 2.20. Pseudocódigo genérico del método SPEA2.

Capítulo 3
El Problema del Cajero Viajante

3.1 Introducción

La *Optimización Combinatoria* [Wikipedia1, Hansen98] (CO – *Combinatorial Optimization*) es una rama de la Optimización en Matemáticas Aplicadas y de la Ciencia de la Computación relacionada a la Investigación de Operaciones y a la Teoría de la Complejidad Computacional, que se dedica al estudio de problemas de optimización que se resuelven mediante la búsqueda de un ordenamiento, selección, agrupación, combinación o permutación de un número finito de elementos discretos, pasos u operaciones según sea el caso, tal que permita obtener el mayor provecho posible.

La importancia de los *Problemas de Optimización Combinatoria* (COP – *Combinatorial Optimization Problems*) radica en que ellos tienen aplicación práctica en diversas áreas, tales como Planificación de Producción, Planificación de Proyectos, Planificación de Recursos, Enrutamiento de Vehículos, Enrutamiento en Telecomunicaciones, Logística, Diseño de Plantas, Diseño de Redes de Telecomunicaciones, entre otros [Yu98].

Los COPs relacionados a casos reales por lo general requieren de soluciones que tengan en cuenta diferentes puntos de vista, lo cual corresponde a la optimización simultánea de diferentes objetivos y que resulta en la aparición de un tipo de COP denominado *Problema Combinatorio de Múltiples Objetivos* (MOCOP – *MultiObjective Combinatorial Optimization Problem*). Los métodos utilizados en la resolución de MOCOPs son estudiados en la rama de la CO denominada *Optimización Combinatoria Multiobjetivo* (MOCO – *MultiObjective Combinatorial Optimization*) [Hansen98].

En su mayoría, los COPs son difíciles de resolver debido a que su espacio de decisión crece de manera exponencial o factorial con relación a su tamaño y a que por lo general no existe un método determinístico que permita su resolución óptima en un tiempo razonable [Wikipedia1]. De esta forma, muchos COPs son clasificados como problemas NP-Duros y/o NP-Completos [Sahni76], que según las definiciones encontradas dentro de la Teoría de la Complejidad Computacional se conoce que la resolución eficiente de estos resultaría en la resolución eficiente de una gran cantidad de problemas del mismo tipo.

El *Problema del Cajero Viajante* (TSP – *Traveling Salesman Problem*), es uno de los COP más estudiados en el campo de la CO, debido a la sencillez de su planteamiento y a su alta complejidad de su resolución, tanto en su versión mono-objetiva como multiobjetiva, ya que pertenece al grupo de problemas NP-Completos. Estas características, sencillez y dificultad, lo han llevado a convertirse en el problema más utilizado para evaluar y comparar los distintos métodos orientados a la resolución de COPs.

Siguiendo la filosofía anterior, en este trabajo se ha empleado al TSP para evaluar la aplicación de los conceptos de la metaheurística de *Optimización de Enjambre de Partículas* (PSO – *Particle Swarm Optimization*) en la resolución de su versión con múltiples objetivos, de manera a presentar un primer avance para la aplicación de dicha metaheurística en la resolución de MOCOPs.

En este capítulo se presenta la formulación matemática del TSP mono-objetivo y multiobjetivo, se citan algunas de sus aplicaciones prácticas, se presentan las bases de la metaheurística de *Optimización de Colonias de Hormigas* (ACO – *Ant Colony Optimization*) y se describen los algoritmos MOACO (*MultiObjective Ant Colony Optimization*) para la resolución de MOCOPs utilizados en el presente trabajo. También se presentan otros métodos para la resolución de TSP multiobjetivo, cuyos resultados disponibles en Internet han sido utilizados como referencia para la comparación de los distintos métodos implementados para este trabajo.

3.2 Formulación del Problema del Cajero Viajante

El TSP puede ser definido como sigue: Sean N ciudades en un territorio dado, totalmente interconectadas entre ellas y con costo de viaje conocido entre cada par de ciudades, se desea encontrar la ruta de menor costo que inicie en cualquier ciudad, pase una sola vez por cada una de las demás ciudades y finalice en la ciudad inicial, de manera a minimizar el costo de recorrido de un agente de comercio que visita las N ciudades.

Cualquier ruta que atraviesa cada ciudad una sola vez y termina en la ciudad de inicio se conoce como *ciclo hamiltoniano* [Lipschutz93]. De esta forma, la solución a un problema TSP dado es el ciclo hamiltoniano menor coste entre sus ciudades.

Una formulación del TSP equivalente a la anterior, en términos de la Teoría de Grafos, es la siguiente:

Definición 3.1 (TSP): En un grafo dado $G = (V, E)$ completamente conexo, donde $V = \{v_1, v_2, \ldots, v_N\}$ es el conjunto de vértices, N es el número de vértices, $E = \{(v_i, v_j) \mid \forall\ v_i, v_j \in V\}$ es el conjunto de aristas y cada arista tiene asociada un nú-

mero real $c_{ij} \geq 0$ que representa el costo de atravesar dicha arista, se debe hallar el ciclo hamiltoniano de costo mínimo a través de los vértices del grafo.

Una solución *S* a un problema TSP puede ser vista como una permutación del conjunto de vértices *V*, la cual induce una selección de *N* elementos del conjunto de aristas *E* que se denota como $S \rightarrow E$. De esta forma, la función de costo para una solución cualquiera *S*, denotada por $f(S)$, se define como:

$$f(S) = \sum c_{ij} \quad \forall (v_i, v_j) \in S \rightarrow E. \tag{3.1}$$

Así, el problema de optimización TSP mono-objetivo se define como:

$$Minimizar \quad f(S). \tag{3.2}$$

En instancias de problemas TSP de *M* objetivos, cada arista tiene asociada un vector de costo $\vec{c}_{ij} \geq 0$ de dimensión *M*, en donde cada componente $1 \leq k \leq M$ de este vector indica el costo de atravesar la arista (v_i, v_j) según el objetivo *k*. De esta forma, la función de costo de una solución *S* según el objetivo *k* viene dada por:

$$f_k(S) = \sum c_{ij}^k \quad \forall (v_i, v_j) \in S \rightarrow E, \tag{3.3}$$

donde c_{ij}^k representa la *k*-ésima componente del vector de costo correspondiente a la arista (v_i, v_j). De esta forma, el problema de optimización TSP de *M* objetivos se define como:

$$Minimizar \quad \vec{f}(S) = [f_1(S), f_2(S), ..., f_M(S)]. \tag{3.4}$$

Aun en su versión mono-objetiva, el TSP es un problema difícil de resolver, ya que para encontrar la solución óptima de una instancia cualquiera es necesaria la exploración exhaustiva de un espacio de búsqueda de tamaño no polinomial. Es decir, un método que garantice en un ciento por ciento la resolución óptima de cualquier instancia TSP debe evaluar todos los ciclos hamiltonianos posibles en la instancia dada para encontrar el ciclo hamiltoniano de menor costo, aunque existen algoritmos que garantizan encontrar en corto tiempo soluciones próximas a la solución óptima sin necesitar explorar todo el espacio de búsqueda [Nilsson03].

El tiempo necesario para evaluar todos los ciclos hamiltonianos de una instancia TSP de *n* ciudades está en relación con el número de ciclos en dicha instancia, el cual viene dado por:

$$\frac{(n-1)!}{2}. \tag{3.5}$$

Como ejemplo, suponiendo que es posible evaluar 1.000.000 de ciclos hamiltonianos por segundo, la tabla siguiente muestra como el tiempo necesario para la evaluación de todos los

ciclos hamiltonianos en instancias TSP crece de manera factorial con respecto al número de ciudades.

Número de ciudades	Cantidad de Ciclos	Tiempo de Resolución
10	181440	< 1s
15	$4{,}4x10^{10}$	12,1 h
20	$6{,}1x10^{16}$	1929 años
50	$3{,}1x10^{62}$	$9{,}7x10^{48}$ años.
100	$4{,}7x10^{155}$	$1{,}5x10^{142}$ años

Tabla 3.1. Tiempo necesario para la evaluación de todos los ciclos hamiltonianos en instancias TSP.

A partir de esto, se puede demostrar que el TSP es un problema del tipo NP (*Non-polynomic deterministic time*), más específicamente del tipo NP-Completo, aunque la prueba de esto escapa al alcance de este trabajo.

Así como en el TSP mono-objetivo, para encontrar el frente Pareto de un problema TSP multiobjetivo es obligatoria la exploración completa de espacio de decisión de manera a obtener la evaluación en todos los objetivos de cada ciclo hamiltoniano, lo cual reduce nuevamente al TSP multiobjetivo en un problema del tipo NP.

3.3 Aplicaciones Prácticas del Problema del Cajero Viajante

Mucho del trabajo sobre el TSP no es motivado por sus aplicaciones directas, sino por el hecho de que el TSP proporciona una plataforma de inicio ideal para el estudio y comparación de los diferentes métodos que se pueden aplicar para la resolución de COPs. Sin embargo, esto no quiere decir que el TSP no tenga aplicaciones prácticas y a continuación se citan algunas de ellas.

3.3.1 Secuenciación de Genomas

Investigadores del *National Institute of Health* de los Estados Unidos, en su tarea de secuenciación de genomas, han formulado el problema de construcción de mapas híbridos de radiación para un genoma como una aplicación directa del TSP [Agarwala00].

El problema consiste en secuenciar una cierta cantidad de mapas de radiación locales de un genoma (pedazos de la información correspondiente al genoma) en un solo mapa híbrido de radiación. En este caso, los mapas locales son considerados como las ciudades y el coste de viaje entre las ciudades está en relación con la probabilidad de que un mapa local dado siga al resto de los demás mapas.

3.3.2 Programa *Starlight*

Un equipo de ingenieros de *Hernández Engineering* de *Houston* y de la Universidad de *Brigham Young*, ha aplicado el TSP para obtener la secuencia de objetos celestiales a ser observados a través de satélites como una propuesta para el programa *Starlight* de la NASA [Bailey00].

La meta es reducir al mínimo el combustible utilizado por los satélites durante las maniobras realizadas para posicionarlos. De esta manera, las ciudades son las posiciones de observación de los distintos objetos celestiales y los costes de viaje están dados por las cantidades de combustible necesarias para posicionar los satélites desde cada una de las posiciones de observación a las demás.

3.3.3 Optimización de Cadenas de Exploración de Chips

La cadena de exploración (*scan chains*) [Boese94, Gupta03] es una ruta incluida en los chips para propósitos de prueba, la cual interconecta en serie todos los elementos del chip. Así, una señal de prueba es introducida en el chip a través de la cadena de exploración, cuya salida permite determinar el buen funcionamiento del chip.

La optimización de las cadenas de exploración conlleva a un mejoramiento del costo y del rendimiento de los chips. De esta forma, la minimización de la longitud del circuito correspondiente a las cadenas de exploración permite (i) disminuir la congestión de alambrado del chip, (ii) reducir el tamaño del chip y (iii) reducir los efectos capacitivos entre los circuitos de prueba y de funcionamiento del chip [Gupta03].

La búsqueda de la mejor forma de disponer las cadenas de exploración entre los distintos elementos del chip deriva en una aplicación directa del TSP como se muestra en [Boese94, Gupta03]; en donde, distintos elementos en el chip son mapeados a las ciudades y las distancias entre las ciudades están en relación con las distancias *Manhattan* [Boese94] entre los elementos del chip. La evaluación de las soluciones puede involucrar a uno o más de los tres puntos citados anteriormente como objetivos a ser optimizados.

3.3.4 Minimización del Recorrido de Brazos Mecánicos

Otra de las aplicaciones prácticas del TSP consiste en encontrar la secuencia de realización de las operaciones de un brazo mecánico que permita minimizar el recorrido de este durante el tratamiento de las piezas. De esta manera, es posible aumentar el número de piezas producidas o tratadas por unidad de tiempo.

Por ejemplo, una empresa griega dedicada a la fabricación de placas de circuitos impresas (PCB – *Printed Circuit Board*s), ha aplicado el TSP con el objetivo de disminuir el tiempo necesario para realizar la perforación de los hoyos en donde irán soldados los distintos componentes de la placa. De esta forma a través del mapeamiento de este proceso en un problema TSP, se pudo obtener una mejor secuencia de perforación que permitió incrementar la taza de perforación de 500 hoyos por hora a 4.800 hoyos por hora [Metelco].

3.3.5 Diseño de Redes en Anillo

El TSP también es aplicado al problema de diseño de redes de comunicación en anillo, en las cuales se necesitan interconectar cierto número de sitios mediante una red de datos con topología de anillo. Esta topología de por si proporciona un mecanismo de reserva en caso de fallas de un solo enlace, puesto que el tráfico se puede reencaminar en la dirección opuesta al fallo.

En este caso, los sitios a interconectar son mapeados a las ciudades y las soluciones pueden ser evaluadas con respecto al costo del tendido de las líneas de interconexión entre los sitios, la confiabilidad de los enlaces, el retardo máximo y promedio en la red, la variación de la taza de transferencia de datos, entre otros.

3.3.6 Problemas de Logística

Las aplicaciones más evidentes del TSP tienen relación con problemas de logística, en los cuales se debe encontrar el camino de menor longitud como para la repartición de periódicos, entrega de correspondencia, recolección de monedas de los teléfonos públicos, recolección de basura, optimización de rutas aéreas, optimización del tráfico en ciudades, entre otros.

En ciertos casos de logística, las soluciones pueden ser evaluadas con respecto a uno o más objetivos si estos no están en función a la longitud de los trayectos o entre ellos, tales como tiempo total de recorrido, costo total de recorrido en dinero, volumen total de carga transportada, etc.

3.4 Optimización de Colonia de Hormigas

La metaheurística de Optimización de Colonia de Hormigas (ACO – *Ant Colony Optimization*), introducida por Dorigo et al. [Dorigo96], es una técnica probabilista de resolución de COPs tales como el TSP, el *Quadraric Asignment Problem* (QAP) [Paciello06], el *Vehicle*

Routing Problem (VRP) [Barán03] y Enrutamiento en Redes de Computadores [Pinto05], inspirada en el comportamiento observado en colonias de hormigas descrito a continuación.

3.4.1 Las Hormigas

Las hormigas son insectos prácticamente ciegos que viven agrupados en colonias y muestran un patrón de interacción social que les permite, entre otras cosas, encontrar el camino más corto entre su nido y los alimentos.

Cuando una hormiga recorre el espacio entre su nido y las fuentes de alimentos, esta deposita a lo largo de su travesía una sustancia química capaz de ser percibida por otras hormigas a la cual se denomina *feromona*.

Al iniciar un recorrido, cada hormiga intenta detectar algún rastro de feromonas depositadas por otras hormigas de su colonia. En caso de no percibir un rastro de feromonas, ella tiende a moverse de manera aleatoria hasta encontrar alimentos o detectar un rastro de feromonas.

Al llegar una hormiga a una fuente de alimentos sin haber seguido ningún rastro de feromonas, ella utiliza su rastro de feromonas para regresar a su nido.

Por otra parte, cuando una hormiga percibe un rastro de feromonas ella tiende a seguirlo. Si la hormiga hubiera descubierto más de un camino de feromonas, ella selecciona uno de esos caminos de manera aleatoria, donde el camino con mayor intensidad de feromonas tiene mayor posibilidad de ser seguido.

A este tipo de interacción colaborativa a través de la modificación del medio físico se denomina *estigmergia* (*stigmergy*), que en el caso de las hormigas resulta en un proceso de establecimiento de caminos por concentración de feromonas.

Un fenómeno complementario del proceso de establecimiento de caminos es la evaporación de las feromonas. De esta forma, los caminos atravesados por cada vez menos hormigas van perdiendo intensidad de feromonas y al dejar de ser atravesados desaparecen por la evaporación total de las feromonas.

Con el transcurrir del tiempo los caminos más cortos tienden a tener mayor intensidad de feromonas dado que son los que se logran recorrer más rápidamente, logrando atraer a la mayoría de las hormigas, la cuales refuerzan nuevamente la cantidad de feromonas en ellos.

La figura 3.1 ilustra este proceso de establecimiento de caminos. En el instante t_0, tres hormigas parten del nido hacia la fuente de alimentos por caminos diferentes. En el instante t_1 se puede observar como las hormigas fueron depositando feromonas (indicadas por las líneas de puntos) a medida que avanzan hacia los alimentos.

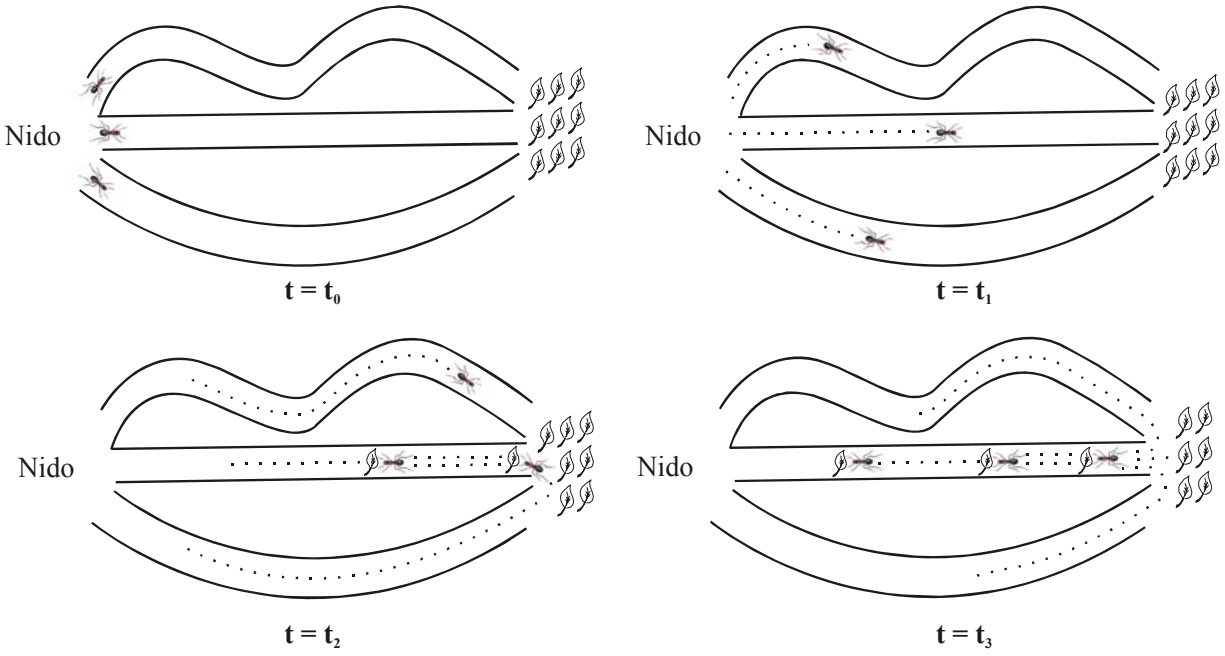


Figura 3.1. Comportamiento de las hormigas reales.

Entre el instante t_1 y t_2, la hormiga que siguió el camino del centro (el más corto) llegó primera a la fuente de alimentos y retornó hacia el nido por el mismo camino debido a que este era el único que contenía feromonas, incrementando nuevamente la cantidad de feromonas en ese trayecto. En el instante t_2, la hormiga que siguió el segundo camino más corto llega a la fuente de alimentos, y decide retornar por el trayecto de la primera hormiga ya que este posee mayor intensidad de feromonas que el camino por el cual ella ha llegado. También se puede notar en este instante que las feromonas depositadas van desapareciendo debido a la evaporación de las mismas. Finalmente en el instante t_3, la tercera hormiga llega a la fuente de alimentos y retorna por el trayecto del medio, debido que en este camino existe una mayor cantidad de feromonas que en los demás caminos.

3.4.2 La Metaheurística de Optimización de la Colonia de Hormigas

La metaheurística ACO [Dorigo96] se inspira en el comportamiento de las hormigas reales arriba descrito. Básicamente, este método consiste es un algoritmo iterativo compuesto por una colonia de hormigas artificiales que recorren las aristas de un grafo de construcción de soluciones, las cuales a medida que avanzan y completan sus recorridos, van depositando cierta cantidad de feromonas artificiales en una tabla de feromonas.

En el grafo de construcción de soluciones, los nodos o vértices representan los posibles estados en los que las hormigas pueden encontrarse durante la construcción de las soluciones,

mientras que las arista del grafo representan los posibles pasos o transiciones entre estados que las hormigas pueden realizar para construir las soluciones. Estas aristas tienen asociadas dos tipos de información que guían a las hormigas, la *información de visibilidad* y la *información de feromonas*.

La información de visibilidad, denotada por $\eta = \{\eta_{ij}\}$, indica la bondad de proseguir con la construcción de una solución desde un nodo *i* hacia un nodo *j* a través de la arista (i, j), la cual se deriva a partir de una valoración conocida a priori sobre la transición del estado *i* al estado *j*, por ejemplo un valor que indica el costo de la transición. Esta información tiene la característica de permanecer constante durante la ejecución del algoritmo.

La información de feromonas, denotada por $\tau = \{\tau_{ij}\}$, indica la calidad de las aristas del grafo en función a la cantidad de feromonas depositadas en la tabla de feromonas. Así, la calidad de la arista (i, j) es proporcional a la cantidad de feromonas depositadas en la entrada τ_{ij} de la tabla de feromonas. Esta información es modificada durante la ejecución del algoritmo, ya sea por adición de feromonas hechas por las hormigas que atraviesan las aristas y/o por sustracción de feromonas debido a la evaporación de las mismas. Inicialmente, todas las entradas válidas de la tabla de feromonas, y por consiguiente todas las aristas de grafo de construcción de soluciones, son inicializadas a un cierto nivel de feromonas τ_0, el cual es definido a priori.

Cada hormiga inicia su recorrido en algún nodo del grafo de construcción correspondiente a un estado inicial, pasando luego a través de los vértices del grafo correspondientes a estados siguientes válidos hasta llegar a un estado final, terminando así la construcción de una solución.

Cuando una hormiga se encuentra en cierto estado *i* no final, ella debe continuar con su recorrido a través de una de las aristas que parten del nodo correspondiente a dicho estado hacia los nodos correspondientes a estados *j* válidos. La selección del siguiente estado, y por consecuencia de una de las artistas que parten del nodo actual, se realiza de manera aleatoria según el procedimiento de la figura 3.2 (también conocido como *algoritmo de la ruleta*) y la distribución de probabilidades dada por [Dorigo96, Dorigo97]:

$$p_{ij} = \begin{cases} \dfrac{(\tau_{ij})^{\alpha} \times (\eta_{ij})^{\beta}}{\sum_{k \in Next(i)} (\tau_{ik})^{\alpha} \times (\eta_{ik})^{\beta}} & \text{si } j \in Next(i) \\ 0 & \text{en caso contrario} \end{cases} \tag{3.6}$$

donde *Next*(*i*) es una función que devuelve el conjunto de estados siguientes válidos para el estado *i*. Los parámetros α y β son definidos a priori e indican la importancia relativa entre la información de feromonas y de visibilidad respectivamente. Mientras que, τ_{ij} y η_{ij} son los valores del nivel de feromonas actual y de visibilidad correspondientes a la arista (*i*, *j*) respectivamente, y p_{ij} es valor de la probabilidad de selección de la arista (*i*, *j*).

```
1.  Leer estado actual i;
2.  Calcular distribución de probabilidades para el estado i.
3.  MAX := 1000;
4.  r := rand(0, MAX); // calcular un número aleatorio 0 < r < MAX.
5.  FOR cada estado j en Next(i) DO
6.     r := r – MAX * pij;
7.     IF (r <= 0) THEN;
8.        retornar j;  // el procedimiento siempre debe retornar aquí.
9.     END_IF
10. END_FOR
```

Figura 3.2. Procedimiento para la selección del siguiente estado *j* a visitar por una hormiga.

Al momento en que una hormiga termina de construir una solución [Dorigo96, Dorigo97] o durante la construcción de la misma [Dorigo97], ella actualiza la tabla de feromonas, de manera a introducir nueva información que guiará a las siguientes hormigas. Otro tipo de actualización de la tabla de feromonas corresponde la evaporación de las mismas, en la cual se disminuye la cantidad de feromonas en las entradas de la tabla en cierta proporción ρ. De esta forma, la actualización de las entradas de la tabla de feromonas, tanto por evaporación como por depósito de feromonas, se realiza de acuerdo con:

$$\tau_{ij} = (1-\rho)\times\tau_{ij} + \rho\times\Delta\tau \tag{3.7}$$

donde $\Delta\tau$ es igual a τ_0 en caso de que la actualización de la tabla se lleve a cabo durante la construcción de las soluciones (*actualización paso a paso* o *actualización local* [Dorigo97]) y en caso de realizarse al final de la construcción de las soluciones (*actualización global* [Dorigo96]) $\Delta\tau$ se calcula como:

$$\Delta\tau = \frac{Q}{F(x)}, \tag{3.8}$$

siendo *Q* un parámetro definido a priori y *F*(*x*) la evaluación de la solución *x* que actualiza la tabla.

La figura 3.3 muestra el pseudocódigo del algoritmo ACO genérico. En la literatura pueden encontrarse varios métodos de optimización mono-objetivos basados en los conceptos de la metaheurística ACO tales como el *Ant System* [Dorigo96], el *Ant Colony System* [Dorigo97], el *Max-Min Ant System* [Stutzle00] y el *Ómicron ACO* [Gómez04].

```
1.  Leer_parámetros();
2.  Incializar_tabla_de_feromonas();
3.  DO
4.     FOR i:= 1 TO NRO_HORMIGAS DO
5.        solución[i] := ∅;
6.        solución[i] := solución[i] U cualquier_estado_inicial();
7.        WHILE(solución_no_completa(solución[i])) DO
8.           siguiente := selec_estado_sgte(solución[i]); /* ec.(3.6) */
9.           solución[i] := solución[i] U siguiente;
10.          IF(actualizacion_paso_a_paso == TRUE) THEN
11.             Actualizar_feromonas_paso_a_paso(); /* según ec. (3.7) */
12.          END_IF
13.       END_WHILE
14.       Evaluar_solución(solución[i]);
15.       Actualizar_mejor_solución();  /* guardar la mejor solución */
16.       Actualizar_feromonas(); /* según ec.(3.7) y (3.8) */
17.    END_FOR
18. WHILE(condición_de_parada() == FALSE)
19. Retornar_mejor_solución();
```

Figura 3.3. Pseudocódigo del algoritmo ACO genérico.

3.5 Optimización de Colonia de Hormigas en Problemas Multiobjetivos

Esta sección se presenta una breve descripción de los métodos ACO para la resolución MOPs (MOACO – *MultiObjective Ant Colony Optimization*) utilizados en este trabajo para resolver el TSP bi-objetivo.

3.5.1 Multiobjective Ant Colony System

El método *MultiObjective Ant Colony System* (MOACS), propuesto por Barán y Schaerer [Barán03], extiende el método *Ant Colony System* (ACS) [Dorigo97] para el tratamiento de MOCOPs. Este método considera la utilización de una tabla de feromonas y M informaciones de visibilidad para resolver un MOCOP de M objetivos.

Una regla de transición de estados pseudoaleatoria es utilizada para seleccionar el siguiente estado j a visitar desde un estado i por una hormiga h, la cual se define como:

$$j = \begin{cases} \underset{k \in Next(i)}{max\_ind} \left(\left[\tau_{ik}\right]^{\alpha} \times \prod_{m=1}^{M} \left[\eta_{ik}^{m}\right]^{\lambda_m^h \beta} \right) & \text{si } q < q_0 \\ S(i) & \text{en caso contrario} \end{cases} \tag{3.9}$$

donde q es un número aleatorio en el intervalo [0, 1], $0 \le q_0 \le 1$es el parámetro que determina el balance entre explotación del conocimiento almacenado en la tabla de feromonas versus la exploración aleatoria de nuevos caminos, τ_{ik} es la información de feromonas correspondiente a la arista (i, k), η_{ik}^{m} es el valor de visibilidad para la arista (i, k) según el objetivo m, los parámetros α y β son los pesos de influencia entre las informaciones de feromonas y de visibili-

dad respectivamente, los parámetros λ_m^h definen la importancia relativa entre las informaciones de visibilidad para la hormiga *h*, la función *max_ind*() retorna el valor del índice *k* que maximiza la expresión que recibe como parámetro y la función *S*(*i*) determina aleatoriamente un estado *j* por el cual continuar la construcción de soluciones de acuerdo con el procedimiento de la figura 3.2 y la distribución de probabilidades dada por:

$$p_{ij} = \begin{cases} \dfrac{\left[\tau_{ij}\right]^{\alpha} \times \prod_{m=1}^{M} \left[\eta_{ij}^{m}\right]^{\lambda_m^h \beta}}{\sum_{k \in Next(i)} \left[\tau_{ik}\right]^{\alpha} \times \prod_{m=1}^{M} \left[\eta_{ij}^{m}\right]^{\lambda_m^h \beta}} & \text{si } j \in Next(i) \\ 0 & \text{en caso contrario.} \end{cases} \tag{3.10}$$

Durante la construcción de soluciones, cada vez que una hormiga se mueve de un estado *i* a otro estado *j*, realiza una actualización local en la entrada (*i*, *j*) de la tabla de feromonas conforme a la ecuación (3.7) con $\Delta\tau = \tau_0$.

Este método utiliza un repositorio para almacenar todas las soluciones no dominadas entre si que se han encontrado. Cuando se encuentra una nueva solución no dominada por los elementos del repositorio, esta es añadida eliminándose las soluciones que resulten dominadas.

Luego de que se haya insertado una nueva solución al repositorio, todas las entradas de la tabla de feromonas son reinicializadas a τ_0, con el objeto de evitar la utilización de la información correspondiente a las soluciones dominadas por la nueva solución [Barán03].

```
1.  Leer_parámetros();
2.  Incializar_tabla_de_feromonas();
3.  R := ∅; // Repositorio de soluciones Pareto.
4.  DO
5.     FOR i:= 1 TO NRO_HORMIGAS DO
6.        solución[i] := ∅;
7.        solución[i] := solución[i] U cualquier_estado_inicial();
8.        WHILE(solución_no_completa(solución[i])) DO
9.           siguiente := selec_estado_sgte(solución[i]); /* ec. (3.9) */
10.          solución[i] := solución[i] U siguiente;
11.          actualizar_feromonas_paso_a_paso(); /* según ec. (3.7) */
12.       END_WHILE
13.       Evaluar_solución(solución[i]); /* según la función (2.1) */
14.       IF (R NO_DOMINA_A solución[i]) THEN
15.          Agregar solución[i] a R;
16.          Eliminar soluciones dominadas de R;
17.          Reinicializar_tabla_de_feromonas();
18.       ELSE
19.          actualizar_feromonas(); /* según ec. (3.7) y (3.11) */
20.       END_IF
21.    END_FOR
22. WHILE(condición_de_parada() == FALSE)
23. Retornar_soluciones_no_dominadas();
```

Figura 3.4 Pseudocódigo genérico del método MOACS.

Por otra parte, si la solución encontrada resulta dominada se actualiza la tabla de feromonas según la ecuación (3.7), donde el valor para $\Delta\tau$ se obtiene según la siguiente ecuación:

$$\Delta\tau^k = \frac{1}{\sum_{i=1}^{M} Norm(f_i(k))}, \tag{3.11}$$

siendo *Norm*() una función que devuelve el valor normalizado de la evaluación de la función objetivo, el cual se halla dividiendo el resultado de la evaluación de la función objetivo por un valor máximo definido a priori para cada objetivo.

El pseudocódigo genérico para el método MOACS se muestra en la figura 3.4.

3.5.2 Multiobjective Max-Min Ant System

El método *Multiobjective Max-Min Ant System* (M3AS), propuesto por Pinto et al. [Pinto05], extiende el *Max-Min Ant System* propuesto por Stutzle et al. [Stutzle00] para la resolución de MOCOPs. Al igual que el método MOACS, este método utiliza un valor de visibilidad para cada objetivo a optimizar y una única tabla de feromonas para almacenar la información de feromonas.

En este método, los valores en las entradas de la tabla de feromonas actualizadas por las soluciones construidas se mantienen entre un valor máximo y un valor mínimo dados según las ecuaciones siguientes:

$$\tau_{MAX} = \frac{\Delta\tau^k}{1-\rho}, \tag{3.12}$$

$$\tau_{MIN} = \frac{\Delta\tau^k}{2 \times m \times (1-\rho)}, \tag{3.13}$$

donde m es el número de hormigas, ρ es el coeficiente de evaporación de feromonas y $\Delta\tau^k$ es la cantidad de feromonas depositada por la hormiga que construyó la solución k, que se calcula según la ecuación (3.11).

El hecho de mantener los valores de las entradas de la tabla de feromonas acotados a niveles máximos y mínimos permite evitar que ciertos enlaces se vuelvan absolutamente más deseables que otros por poseer un alto nivel de feromonas, mientras que otros enlaces resulten no elegibles por tener un bajo nivel de feromonas. De esta forma, se pretende mantener un balance entre la explotación de la información almacenada en la tabla de feromonas y la exploración de nuevas áreas del espacio de búsqueda [Stutzle00].

Para la construcción de soluciones, el siguiente estado a visitar por una hormiga se determina según el procedimiento de la figura 3.2 y la distribución de probabilidades dada por la ecuación (3.10). También al pasar de un estado a otro, se realiza una actualización local de la tabla de feromonas conforme a la ecuación (3.7), con $\Delta\tau = \tau_0$.

El método M3AS utiliza un repositorio de soluciones no dominadas. Cada vez que se encuentra una solución no dominada por las soluciones almacenadas en el repositorio, esta es insertada removiéndose todas las soluciones que resulten dominadas.

La tabla de feromonas se actualiza nuevamente luego de que todas las hormigas han construido una solución. Primeramente, se evalúa la evaporación de las feromonas en todas las entradas (i, j) de la tabla de feromonas según:

$$\tau_{ij} = (1-\rho)\times\tau_{ij}, \tag{3.14}$$

y luego se realiza la adición de feromonas en todas las entradas de la tabla indicadas por cada una de las soluciones del repositorio, manteniendo los valores de feromonas en estas entradas entre los valores máximo y mínimo correspondientes.

El pseudocódigo genérico del método M3AS se presenta en la figura que sigue.

```
1.  Leer_parámetros(); Incializar_tabla_de_feromonas();
2.  R := ∅; // Repositorio de soluciones Pareto.
3.  DO
4.     FOR i:= 1 TO NRO_HORMIGAS DO
5.        solución[i] := ∅;
6.        solución[i] := solución[i] U cualquier_estado_inicial();
7.        WHILE(solución_no_completa(solución[i])) DO
8.           siguiente := selec_estado_sgte(solución[i]); /* ec.(3.10) */
9.           solución[i] := solución[i] U siguiente;
10.          Actualizar_feromonas_paso_a_paso(); /* según ec. (3.7) */
11.       END_WHILE
12.       Evaluar_solución(solución[i]); /* según la función (2.1) */
13.       IF (R NO_DOMINA_A solución[i]) THEN
14.          Agregar solución[i] a R;
15.          Eliminar soluciones dominadas de R;
16.       END_IF
17.    END_FOR
18.    Evaporar_feromonas(); /* según ec. (3.14) */
19.    FOR k:=1 TO tamaño(R) DO
20.       calcular Δτ^k, τ_MAX y τ_MIN para R[k]; /* ec.(3.11)(3.12)(3.13) */
21.       FOR cada entrada (i,j) en R[k] DO
22.          τ_ij := τ_ij + Δτ^k;
23.          IF(τ_ij > τ_MAX) THEN
24.             τ_ij := τ_MAX;
25.          ELSE IF(τ_ij < τ_MIN) THEN
26.             τ_ij := τ_MIN;
27.          END_IF
28.       END_FOR
29.    END_FOR
30. WHILE(condición_de_parada() == FALSE)
31. Retornar_soluciones_no_dominadas();
```

Figura 3.5. Pseudocódigo genérico del método M-MMAS.

3.5.3 Pareto Ant Colony Optimization

El método *Pareto Ant Colony Optimization* (PACO), propuesto por Doerner et al. [Doerner02], extiende el método ACS [Dorigo97] para la resolución de MOCOPs mediante la utilización de una tabla de feromona por cada objetivo a optimizar. En este método, la actuali-

zación global de las tablas de feromonas se realiza en cada iteración por las dos mejores hormigas en cada objetivo.

En cada iteración, cada hormiga calcula un conjuntos de pesos $\vec{w} = [w_1, w_2, ..., w_M]$ con $0 \le w_i \le 1$, que los utiliza de manera a combinar linealmente la información de los trayectos de feromonas. Cuando una hormiga en un estado *i* debe seleccionar el siguiente estado *j* a visitar, ella utiliza la siguiente regla de transición:

$$j = \begin{cases} \underset{k \in Next(i)}{max\_ind} \left(\left[\sum_{m=1}^{M} w_m \times \tau_{ik}^m \right]^\alpha \times [\eta_{ik}]^\beta \right) & \text{si } q < q_0 \\ S(i) & \text{en caso contrario} \end{cases} \tag{3.15}$$

donde *M* es el número de objetivos, η_{ik} es el valor resultante de la agregación de las informaciones de visibilidad correspondientes a la arista (*i*, *k*), τ_{ik}^m es la información de feromonas correspondiente a la arista (*i*, *k*) en la tabla *m*, la función *max_ind*() retorna el valor del índice *k* que maximiza la expresión que recibe como parámetro y la función *S*(*i*) determina al azar un siguiente estado *j* a visitar de acuerdo con el procedimiento de la figura 3.2 y la distribución de probabilidades dada por:

$$p_{ij} = \begin{cases} \dfrac{\left[\sum_{m=1}^{M} w_m \times \tau_{ij}^m \right]^\alpha \times [\eta_{ij}]^\beta}{\sum_{k \in Next(i)} \left[\sum_{m=1}^{M} w_m \times \tau_{ik}^m \right]^\alpha \times [\eta_{ik}]^\beta} & \text{si } j \in Next(i) \\ 0 & \text{en caso contrario.} \end{cases} \tag{3.16}$$

Cada vez que una hormiga atraviesa una arista (i, j), esta realiza una actualización local en cada tabla *m* de feromonas de acuerdo a:

$$\tau_{ij}^m = (1 - \rho) \times \tau_{ij}^k + \rho \times \tau_0 \qquad \forall m \in \{1, ..., M\}. \tag{3.17}$$

Luego de que cada hormiga haya construido una solución, la actualización global de cada tabla *m* de feromonas se realiza utilizando las dos mejores soluciones en el objetivo *m* conforme con:

$$\tau_{ij}^k = (1 - \rho) \times \tau_{ij}^k + \rho \times \Delta\tau \tag{3.18}$$

donde el valor para $\Delta\tau$ está dado por:

$$\Delta\tau = \begin{cases} 15 & \text{si la arista } (i, j) \text{ está presente en las dos mejores soluciones} \\ 10 & \text{si la arista } (i, j) \text{ está presente en la mejor solución} \\ 5 & \text{si la arista } (i, j) \text{ está presente en la segunda mejor solución} \\ 0 & \text{en caso contrario.} \end{cases} \tag{3.19}$$

Al igual que en los métodos anteriores, las soluciones no dominadas encontradas durante el proceso son almacenadas en un repositorio de soluciones no dominadas.

En la figura siguiente se ilustra el pseudocódigo genérico para el método PACO.

```
1.   Leer_parametros(); Incializar_tablas_de_feromonas();
2.   R := ∅; // Repositorio de soluciones Pareto.
3.   DO
4.      FOR i:= 1 to NRO_HORMIGAS DO
5.         solución[i] := ∅;
6.         solución[i] := solución[i] U cualquier_estado_inicial();
7.         WHILE(Solución_no_completa(solución[i])) DO
8.            siguiente := selec_estado_sgte(solución[i]); /* según ec.(3.15) */
9.            solución[i] := solución[i] U siguiente;
10.           Actualizar_tablas_de_feromonas_paso_a_paso(); /* según ec. (3.17) */
11.        END_WHILE
12.        Evaluar_solución(solución[i]); /* según la función (2.1) */
13.        IF (R NO_DOMINA_A solución[i]) THEN
14.           Agregar solución[i] a R;
15.           eliminar soluciones dominadas de R;
16.        END_IF
17.     END_FOR
18.     Actualizar_tablas_de_feromonas() /* según ec. (3.18) y (3.19) */
19.  WHILE(condición_de_parada() == FALSE)
20.  Retornar_soluciones_no_dominadas();
```

Figura 3.6. Pseudocódigo genérico del método PACO.

3.6 Otros Trabajos Acerca del Problema del Cajero Viajante Multiobjetivo

3.6.1 Multiobjective Genetic Local Search

El método *MultiObjective Genetic Local Search* (MOGLS), propuesto por Jaszkiewicz [Jaszkiewicz02], se basa en la idea de que buscar todas soluciones Pareto óptimas de un MOP es igual a encontrar el óptimo de todas las posibles funciones de agregación ponderada de los objetivos (véase sección 2.5.1), en donde estas funciones se diferencian una de otra solamente en el valor de ponderación asignado a cada objetivo. Además, el método MOGLS es considerado como el más eficiente para la resolución del TSP multiobjetivo [Paquete03, Paquete04], esto se refiere a que este método consigue soluciones de muy buena calidad en un corto tiempo de ejecución, según lo reportado por Jaszkiewicz [Jaszkiewicz02].

El método MOGLS implementa la optimización de todas las posibles funciones de agregación ponderada mediante la selección aleatoria de un vector de pesos en cada iteración.

Cada iteración del método MOGLS consiste en la recombinación genética de dos soluciones cuya descendencia es utilizada como solución inicial de un procedimiento de búsqueda local que optimiza la función de agregación ponderada generada de manera aleatoria [Jaszkiewicz02].

El pseudocódigo genérico del método MOGLS se muestra en la figura 3.7. Inicialmente se reciben los valores para los parámetros del algoritmo, *K* y *S*, que indican el tamaño de la población temporal *TP* y el número de soluciones iniciales respectivamente. Luego, se procede a la generación iterativa de *S* soluciones iniciales, en donde cada una de estas soluciones es optimizada a través de un procedimiento de búsqueda local según una función de agregación ponderada obtenida a través de generación aleatoria de un vector de pesos $\vec{W}$ conforme a:

$$\vec{W} = \begin{bmatrix} w_1 = 1 - \sqrt[M-1]{rand\ (\ )} \\ \vdots \\ w_j = \left(1 - \sum_{i=1}^{j-1} w_i\right) \times \left(1 - \sqrt[M-1-j]{rand\ (\ )}\right) \\ \vdots \\ w_M = 1 - \sum_{i=1}^{M-1} w_i \end{bmatrix}, \tag{3.20}$$

donde *M* es el número de objetivos del problema y *rand*() es una función que retorna un número aleatorio en el intervalo [0, 1].

```
1.  Recibir K // Tamaño de la población temporal TP
2.  Recibir S // Número de soluciones iniciales
3.  PE := ∅;
4.  CS := ∅;
5.  FOR i := 1 TO S DO
6.     Generar una función de agregación ponderada aleatoria F;
7.     Generar una solución inicial x;
8.     Optimizar localmente x según F;
9.     Agregar x a CS;
10.    Agregar x a PE si es no dominada y eliminar soluciones dominadas;
11. END_FOR
12. DO
13.    Generar una función de agregación ponderada aleatoria F;
14.    TP := Seleccionar las K mejore soluciones en CS para F;
15.    Elegir al azar dos soluciones x1 y x2 de TP;
16.    x3 := Recombinar soluciones x1 y x2;
17.    Optimizar localmente x3 según F;
18.    IF (x3 es mejor que la peor solución en TP y no esta en TP) THEN
19.       Agregar x3 a CS;
20.    END_IF
21.    Agregar x3 a PE si es no dominada y eliminar soluciones dominadas;
22. WHILE(condición_de_parada() = FALSE)
23. Retornar soluciones en PE;
```

Figura 3.7. Pseudocódigo del método MOGLS.

Cada solución inicial es optimizada y agregada al conjunto de reproducción *CS* (*Current Solution Set*), y de ser una solución no dominada es agregada al repositorio *PE* (*Potential Efficient solutions*). El conjunto de reproducción *CS* es de tamaño igual a *K* por *S* y es gestionado como una cola de orden LIFO (*Last In, First Out* – El último en entrar es el primero en salir).

Posteriormente se siguen con las iteraciones del método, en las cuales se halla una nueva función de agregación ponderada aleatoria según la ecuación (3.20) y se seleccionan las *K* mejores soluciones dentro de *CS* para la función generada, las cuales son almacenadas en *TP*. De *TP* se seleccionan con probabilidad uniforme dos soluciones que son recombinadas para obtener una nueva solución la cual es optimizada localmente según una función de agregación ponderada generada aleatoriamente. Si la solución optimizada resulta mejor que la peor solución en *TP* en términos de la función de agregación y es diferente en el espacio de decisión a las soluciones en *TP*, esta es agregada a *CS*. Por otra parte, si la solución optimizada resulta ser no dominada por los elementos de *PE*, esta es agregada a *PE* eliminándose todas las soluciones que resulten dominadas. Las iteraciones concluyen una vez que se cumpla una condición de parada, como un número máximo de iteraciones o un tiempo máximo de ejecución. El método finaliza retornando todas las soluciones no dominadas almacenadas en *PE*.

Para resolver el TSP multiobjetivo a través del método MOGLS [Jaszkiewicz02] el operador de recombinación utilizado consiste en los siguientes pasos: (i) insertar en la solución hija las aristas comunes en ambos padres y (ii) completar el ciclo hamiltoniano en la solución hija con arcos seleccionados aleatoriamente.

El método de búsqueda local utilizado para resolver el TSP multiobjetivo con el método MOGLS [Jaszkiewicz02], consiste en la aplicación iterativa de la heurística 2-opt (definida en la siguiente sección), que para las soluciones obtenidas mediante el operador de recombinación se realiza en dos fases, en la primera fase no se tienen en cuenta las aristas comunes de las soluciones padres para el intercambio de arcos y en la segunda fase todas las aristas son consideradas para el intercambio de arcos. También, la optimización local de las soluciones iniciales se realiza aplicando el primer intercambio de arcos que represente una mejora de la solución actual, mientras que la optimización local de las soluciones obtenidas por recombinación, se realiza buscando y aplicando el intercambio de arcos que mejor optimiza la solución actual. En ambos casos, la búsqueda local finaliza cuando probados todos los posibles intercambios sobre la solución actual no se encuentra una mejora.

3.6.2 Pareto Local Search

Paquete et al. [Paquete04] presentan un método denominado *Pareto Local Search* (PLS) que extiende las heurísticas 2-opt, 2h-opt y 3-opt del TSP mono-objetivo [Nilson03] para generar el frente Pareto de instancias de problemas TSP multiobjetivo.

La heurística 2-opt consiste en generar a partir de un ciclo hamiltoniano inicial otro ciclo hamiltoniano que se diferencie en 2 aristas del mismo. La heurística 2h-opt consiste en la ge-

neración de un ciclo hamiltoniano que se diferencia en 2 aristas del ciclo inicial y el movimiento aleatorio de una ciudad, elegida también aleatoriamente, de una posición a otra dentro del ciclo generado. La heurística 3-opt consiste en la generación de un ciclo hamiltoniano que se diferencia en 3 aristas del ciclo inicial. Al utilizar una de estas heurísticas para hallar todas las posibles variaciones de un ciclo inicial dado, se dice que se ha generado una *vecindad de soluciones* para este ciclo.

El pseudocódigo del método PLS es ilustrado en la figura 3.8. Este método inicia con la generación aleatoria de una solución inicial, la cual es añadida dentro del repositorio de soluciones no dominadas. Luego en cada iteración se selecciona al azar una solución no marcada del repositorio de soluciones no dominadas y se genera su vecindad de soluciones aplicando una de las heurísticas mencionadas anteriormente. Todas las soluciones en la vecindad de la solución seleccionada que resultan no dominadas por los elementos del repositorio son insertadas en este, eliminándose del mismo todas las soluciones que resulten dominadas. Al final de cada iteración se marca la solución seleccionada del repositorio y el procedimiento se repite hasta que todas las soluciones en el repositorio queden marcadas.

```
1.  A := ∅; // Repositorio de soluciones no dominadas
2.  s := Generar_solución_inicial();
3.  Agregar_al_repositorio(A, s);
4.  DO
5.     s := Seleccionar_solución(A);
6.     V := Generar_vecindad(s); // a través de 2-opt, 2h-opt o 3-opt
7.     FOR s' en V DO
8.        IF (s' NO_DOMINADO_POR r, ∀r ∈ R) THEN
9.           Agregar_al_repositorio(A, s');
10.          Eliminar_soluciones_dominadas(A, s');
11.       END_IF
12.    END_FOR
13.    Marcar s;
14.  WHILE (existen soluciones sin marcar en A);
15.  Retornar soluciones en A;
```

Figura 3.8. Pseudocódigo del método PLS.

De manera general, el método PLS obtiene una aproximación al frente Pareto de una instancia TSP multiobjetiva a partir de la generación y combinación de los frentes Pareto locales en las vecindades de las soluciones no dominadas encontradas, obteniendo estas vecindades a través de la aplicación de una de las heurísticas 2-opt, 2h-opt o 3-opt del TSP mono-objetivo. El método PLS [Paquete03] en todas sus versiones, 2-opt, 2h-opt y 3-opt fue testado contra el método MOGLS [Jaszkiewicz02] en 7 instancias TSP bi-objetivas, que son: KROAB100, KROAC100, KROAD100, KROBC100, KROBD100 y KROCD100 de 100 ciudades cada una y KROAB150 de 150 ciudades.

Los resultados arrojados por cada versión del método PLS y del método MOGLS fueron comparados de acuerdo a una métrica de calidad y a una métrica de cobertura (dominancia

Pareto) [Paquete04]. Estas evaluaciones indican que las versiones 2-opt PLS y 2h-opt PLS presentan los peores resultados, siendo la versión 3-opt PLS muy ligeramente superior al método MOGLS en términos de calidad de soluciones, pero a costa de tomar un tiempo de ejecución considerablemente mayor para conseguir dicha ventaja [Paquete04].

Finalmente, Paquete et al. [Paquete04] concluyen que el método PLS mantiene un alto nivel de simplicidad debido a que no utiliza un punto ideal para aproximar las soluciones, no requiere una agregación lineal de los objetivos y no necesita de un establecimiento previo de parámetros. Sin embargo, el inconveniente del PLS es que requiere de un tiempo de computación mayor en comparación a otros métodos para conseguir buenos resultados, como ocurre al compararlo con el método MOGLS, según lo expresado en el párrafo anterior.

Aunque el método PLS [Paquete04] resulta específico para el TSP, su lógica puede ser extendida a otros MOPs, utilizando una heurística y/o estrategia de búsqueda local y un método de generación de soluciones iniciales propias de cada MOP que se pretenda resolver.

3.6.3 Two Phase Local Search

Paquete et al. [Paquete03] presentan un método denominado *Two Phase Local Search* (TPLS) para aproximar el conjunto Pareto de instancias TSP bi-objetivas, motivado principalmente por un enfoque que explota la efectividad de las heurísticas de búsqueda local.

Como su nombre lo indica, el método TPLS se divide en dos fases. La primera consiste es encontrar una solución óptima considerando un solo objetivo, usando para tal efecto un algoritmo mono-objetivo efectivo o una técnica heurística. Esta primera fase provee entonces la solución inicial para la búsqueda de soluciones no dominadas en la segunda fase, en la cual se utiliza una secuencia de diferentes funciones de agregación ponderada (véase sección 2.5.1) y un algoritmo de búsqueda local, las cuales son aplicadas iterativa y encadenadamente, es decir, la búsqueda local del óptimo de la función de agregación ponderada actual inicia con el óptimo encontrado para la función de agregación ponderada anterior.

El pseudocódigo del método TPLS es ilustrado en la figura 3.9. Este método inicia con la recepción del conjunto de funciones de ponderación W a aplicar y la generación de una solución óptima considerando un solo objetivo a través del procedimiento Generar_solución_inicial() el cual utiliza por debajo una heurística de búsqueda local o un algoritmo exacto efectivo para resolver la versión de mono-objetiva del MOP, obteniéndose la solución óptima s_0 con la cual se inicializa el archivo de soluciones A. Todo lo anterior constituye la primera fase del método.

```
1.  Recibir el conjunto de funciones de Ponderación W;
2.  /* Primera Fase */
3.  A := ∅; // archivo de soluciones
4.  s0 := Generar_solución_inicial(); // considerando un solo objetivo
5.  Actualizar_archivo(A, s0)
6.  /* Segunda Fase */
7.  FOR i := 1 TO |W| DO
8.      si := Búsqueda_local(si-1, Wi);
9.      Actualizar_archivo(A, si);
10. END_FOR
11. Filtrar_soluciones(A);
12. Retornar soluciones no dominadas en A;
```

Figura 3.9. Pseudocódigo del método TPLS.

La segunda fase de método consiste en lo siguiente. Realizar para cada función de agregación W_i a ser aplicada al MOP un procedimiento de búsqueda local utilizando como solución inicial la solución óptima encontrada en la iteración anterior s_{i-1} y luego actualizar el conjunto de soluciones A con el óptimo encontrado para tal función de agregación. Una vez que todas las funciones de agregación han sido exploradas por el procedimiento de búsqueda local, se filtra el archivo A eliminando las soluciones dominadas contenidas en el y retornando todas las soluciones no dominadas.

Una ventaja obvia del método TPLS [Paquete03] es su modularidad, ya que en las operaciones Generar_solución_inicial() y Búsqueda_local() varios tipos de algoritmos y/o heurísticas pueden ser aplicados subyacentemente, incluso pudiendo ser aplicada la misma técnica en ambas operaciones.

Para el caso específico del TSP bi-objetivo Paquete et al. [Paquete03] han aplicado en la primera fase, a través de la operación Generar_solución_inicial(), un algoritmo de *búsqueda local iterativa* (ILS – *Iterative Local Search*) denominado *3.first.ils*, que consiste en lo siguiente:

1- Generar aleatoriamente una solución inicial y establecerla como solución actual.
2- Generar una solución tentativa aplicando la heurística 3-opt sobre la solución actual.
3- Actualizar la solución actual con la solución tentativa si esta última es mejor.
4- Si la solución actual no ha cambiado durante cierta cantidad de iteraciones, específicamente durante 50, 100, y 150 iteraciones para problemas de 100, 150 y 200 ciudades respectivamente, utilizar un *método de perturbación* para escapar del óptimo local, el cual consiste en cortar la solución actual en 4 pedazos y reconectarlos de manera diferente a la original.
5- Repetir los pasos 2 a 4 durante cierta cantidad de iteraciones.

Para la segunda fase, a través de la operación Búsqueda_local(), las siguientes estrategias fueron testadas [Paquete03]:

- *2.best*, que consiste en retornar a la solución que mejor optimiza la función de agregación actual dentro de la vecindad de soluciones de la solución inicial generada por la heurística 2-opt (véase sección 3.6.2).
- *3.best*, equivalente a la estrategia 2.best pero utilizando la heurística 3-opt.
- *2.first,* que consiste en retornar a la primera solución que optimice la función de agregación actual en la vecindad de soluciones de la solución inicial obtenida aplicando la heurística 2-opt.
- *3.first*, equivalente a la estrategia 2.first pero utilizando la heurística 3-opt.
- *2.first.ils*, equivalente al algoritmo de búsqueda local iterativa 3.first.ils pero usando la heurística 2-opt.
- *3.first.ils*, algoritmo de búsqueda local iterativa utilizado para generar las soluciones iniciales.
- *2.best.ils*, equivalente al algoritmo 2.first.ils pero actualizando la solución actual en cada iteración con la mejor solución en su vecindad de soluciones generada por la heurística 2-opt.
- *3.best.ils*, equivalente al algoritmo 2.best.ils pero utilizando la heurística 3-opt.

Para las distintas estrategias utilizadas en la segunda fase en el método TPLS, Paquete et al. [Paquete03] encontraron que la versión 3.first.ils produce mejores soluciones que las demás estrategias en las instancias TSP bi-objetivas KROAB100 y KROAB150 y KROAB200. También notaron que las estrategias que utilizan la heurística 3-opt generan mejores soluciones que las que utilizan la heurística 2-opt, pero requieren de un tiempo de ejecución mayor, siendo la versión que utiliza la estrategia 3.first.ils la que requiere de mayor tiempo de cómputo.

El principal problema de TPLS consiste en que el método inicia la búsqueda de soluciones solo del extremo en el cual se optimiza un solo objetivo [Paquete03]. Para aliviar este problema Paquete et al. [Paquete03] plantean iniciar procedimientos TPLS de manera secuencial e independiente desde todas las soluciones iniciales que optimicen cada uno de los objetivos individualmente, retornando al final todas las soluciones no dominadas que se hallan en la unión de los resultados de todos los procedimientos TPLS. De esta forma, se obtiene un nuevo método de búsqueda denominado *Double Two Phase Local Search* (D-TPLS), cuyo pseudocódigo se ilustra en la figura 3.10.

Para las mismas estrategias utilizadas en la segunda fase, las versiones de D-TPLS superan a sus respectivas versiones TPLS en todas las instancias TSP bi-objetivas anteriores, siendo nuevamente la versión que utiliza la estrategia 3.first.ils la que presenta los mejores resultados en términos de calidad de soluciones, a expensas de tomar mayor tiempo de ejecución [Paquete03].

```
1.  Recibir el conjunto de funciones de Ponderación W;
2.  A := ∅; // archivo de soluciones
3.  FOR k := 1 TO nro_objetivos DO
4.     /* Primera Fase */
5.     s0 := Generar_solución_inicial(k); // considerando el objetivo k
6.     Actualizar_archivo(A, s0)
7.     /* Segunda Fase */
8.     FOR i := 1 TO |W| DO
9.        si := Búsqueda_local(si-1, Wi);
10.       Actualizar_archivo(A, si);
11.    END_FOR
12. END_FOR
13. Filtrar_soluciones(A);
14. Retornar soluciones no dominadas en A;
```

Figura 3.10. Pseudocódigo del método D-TPLS.

TPLS y D-TPLS poseen el siguiente punto bajo, ambos métodos solo consideran las soluciones que optimizan las funciones de agregación, limitando el número máximo posible de soluciones no dominadas retornadas por estos métodos a $|W|$ y $2|W|$ respectivamente [Paquete03].

Para solucionar este problema Paquete et al. [Paquete03] proponen agregar al archivo A las soluciones no dominadas por s_i que se encuentren en su vecindad de soluciones generada a través de la heurística 2-opt. Aplicando esta idea al método D-TPLS, se obtiene un nuevo método denominado *Pareto Double Two Phase Local Search* (PD-TPLS) [Paquete03]. El pseudocódigo para el método PD-TPLS es ilustrado en la figura 3.11.

```
1.  Recibir el conjunto de funciones de Ponderación W;
2.  A := ∅; // archivo de soluciones
3.  FOR k := 1 TO nro_objetivos DO
4.     /* Primera Fase */
5.     s0 := Generar_solución_inicial(k); // considerando el objetivo k
6.     Actualizar_archivo(A, s0)
7.     /* Segunda Fase */
8.     FOR i := 1 TO |W| DO
9.        si := Búsqueda_local(si-1, Wi);
10.       Actualizar_archivo(A, si);
11.       Vi = Generar_vecindad_2opt(si);
12.       FOR cada v ∈ Vi DO
13.          IF (si NO_DOMINA_A v) THEN
14.             Actualizar_archivo(A, v);
15.          END_IF
16.       END_FOR
17.    END_FOR
18. END_FOR
19. Filtrar_soluciones(A);
20. Retornar soluciones no dominadas en A;
```

Figura 3.11. Pseudocódigo del método PD-TPLS.

Nuevamente todas las variantes de estrategias de búsqueda local en la segunda fase fueron probadas para PD-TPLS, resultando otra vez mejor en términos de calidad de soluciones aquella versión que utiliza la estrategia 3.first.ils en la segunda fase y siendo a la vez la versión que toma mayor tiempo de ejecución.

También cada variante PD-TPLS fue comparada con respecto a su versión correspondiente en D-TPLS resultando las primeras mejores en términos de calidad de soluciones y con solo pequeños incrementos en los tiempos de ejecución [Paquete03].

Capítulo 4
Optimización de Enjambre de Partículas

4.1 Introducción

En la naturaleza algunas especies de aves, peces e insectos coexisten en grupos dentro de los cuales puede observarse un comportamiento social que permite a un individuo comunicar su suceso a los demás componentes del grupo, lo cual resulta en un proceso colectivo que ayuda a todos los individuos a satisfacer de la mejor manera posible sus necesidades más inmediatas.

Las aves por ejemplo, realizan una exploración en grupo de la zona en que se encuentran, durante la cual cada una trata de mantenerse próxima a las demás [Reynolds87]. Así, cuando cierta ave decide aterrizar o cambiar de dirección por haber detectado alimentos o por otro motivo, ella puede atraer consigo a algunas o a todas las demás transmitiendo de esta forma la información.

Las abejas también conviven en grupos, cuando una abeja obrera detecta alimentos vuelve a su colmena y realiza cierto tipo de danza con lo cual transmite a sus pares la distancia y dirección de la fuente de alimentos [Wikipedia2].

La metaheurística de Optimización de Enjambre de Partículas (PSO – *Particle Swarm Optimization*), desarrollada por Kennedy y Eberhart [Kennedy95] en 1995, se inspira en el comportamiento social de estos grupos de individuos. Esta metaheurística surge a partir de las observaciones realizadas por sus autores sobre sistemas de simulación del movimiento de parvadas de pájaros y bancos de peces propuestos por Reynolds [Reynolds87] y Heppner [Heppner90].

En el modelo de Heppner [Heppner90] se incluye un punto de atracción hacia el cual todos los agentes (aves artificiales) son atraídos. Los agentes inicialmente vuelan sin dirección fija, más tarde empiezan a formar grupos que posteriormente son atraídos hacia el punto de atracción y en el cual todos terminan ubicados luego de un cierto tiempo.

Kennedy y Eberhart [Kennedy95] notaron el hecho de que en este modelo los agentes conocen la ubicación exacta del punto de atracción, mientras que en la realidad las aves no tienen idea del lugar exacto en donde existen alimentos (punto de atracción). Así fue que ellos modificaron el sistema de simulación para que los agentes no tuvieran conocimiento de la posición exacta del punto de atracción, pero que si fuesen capaces de evaluar su posición actual de manera a determinar que tan buena es esta con respecto a la ubicación del punto de

atracción. También dotaron a las aves artificiales con una pequeña memoria que les permite recordar la mejor posición en la que se han encontrado y con la habilidad de conocer la mejor posición encontrada por cada individuo del grupo. De esta forma, los agentes artificiales pueden determinar en base a estas informaciones la dirección a seguir para localizar el punto de atracción sin conocer realmente su ubicación exacta.

Kennedy y Eberhart [Kennedy95] descubrieron que su nuevo modelo de simulación podía ser aplicado como un optimizador para la resolución de problemas de optimización de funciones matemáticas continuas al cual denominaron *Optimización de Enjambre de Partículas* (PSO – *Particle Swarm Optimization*) y que se explica más detalladamente en la siguiente sección.

Luego de la publicación del primer trabajo sobre PSO [Kennedy95], aparecieron varios trabajos que tratan de la aplicación de esta metaheurística en la resolución de problemas de optimización continuos relacionados con casos prácticos, tales como entrenamiento de redes neuronales [Eberhart98], control de voltaje y potencia reactiva en redes eléctricas [Fukuyama01], análisis del tremor humano [Eberhart99], optimización de problemas de ingeniería [Hu03], entre otros.

Para ilustrar el impacto que ha tenido la metaheurística PSO la figura 4.1 muestra la cantidad de trabajos referentes a ella publicados por año desde su aparición en el año 1995 hasta finales del año 2005. Estos datos se han obtenido realizando una búsqueda y conteo de trabajos sobre PSO en sitios Web correspondientes a la librería digital de la IEEE y la ACM (www.computer.org), la librería digital de literatura científica del Colegio de Información Científica y Tecnológica de la Universidad Estatal de Pennsylvania (citeseer.ist.psu.edu) y del sitio Web sobre PSO mantenido el Dr. Xiaohui Hu (www.swarmintelligence.org).

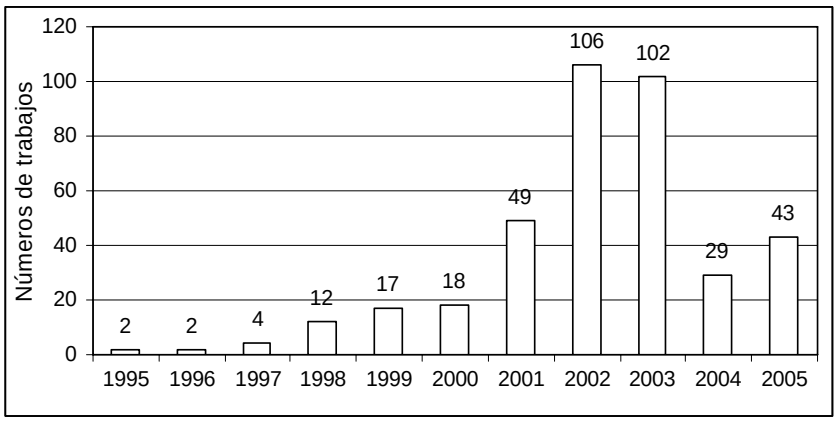


Figura 4.1. Números de trabajos sobre PSO por año.

El resto de este capítulo tratará de los conceptos relacionados a la metaheurística PSO y de su implementación computacional para la resolución de problemas mono-objetivos y multiob-

jetivos. También se presentará una recopilación de algunos métodos propuestos que tratan la aplicación de dicha metaheurística en la resolución de MOPs y la adaptación aplicada al PSO en este trabajo para la resolución del TSP multiobjetivo.

4.2 La Metaheurística de Optimización de Enjambre de Partículas

Básicamente, la metaheurística PSO [Kennedy95] consiste en un algoritmo iterativo basado en una población de individuos denominada *enjambre*, en la que cada individuo denominado *partícula*, se dice que sobrevuela el espacio de decisión en busca de soluciones óptimas.

En un espacio de búsqueda N-dimensional, cada partícula i del enjambre conoce su posición actual $X_i = [x_{i1}, x_{i2},\ldots, x_{iN}]$, la velocidad $V_i = [v_{i1}, v_{i2},\ldots, v_{iN}]$ con la cual ha llegado a dicha posición y la mejor posición $P_i = [p_{i1}, p_{i2},\ldots, p_{iN}]$ en la que se ha encontrado, denominada *mejor posición personal*. Además, todas las partículas conocen la mejor de entre todas las mejores posiciones personales en el enjambre $G = [g_1, g_2,\ldots, g_N]$, a la cual se denomina *mejor posición global*.

En cada iteración t del algoritmo, cada componente j de la velocidad y la posición de cada partícula i del enjambre se actualiza conforme a las siguientes ecuaciones:

$$v_{ij}^{t+1} = \omega \times v_{ij}^{t} + C_1 \times rand(\,) \times \left(p_{ij}^{t} - x_{ij}^{t}\right) + C_2 \times rand(\,) \times \left(g_j^{t} - x_{ij}^{t}\right), \tag{4.1}$$

$$x_{ij}^{t+1} = x_{ij}^{t} + v_{ij}^{t+1}, \tag{4.2}$$

donde ω es el parámetro de inercia, C_1 es el parámetro cognitivo, C_2 es el parámetro social y *rand*() es una función que retorna un número aleatorio en el intervalo [0, 1].

La ecuación (4.1) calcula un nuevo vector de velocidad para la i-ésima partícula a partir de su velocidad actual, la distancia euclidiana a su mejor posición personal y la distancia euclidiana a la mejor posición global. En la ecuación (4.2) las componentes del vector de posición de la i-ésima partícula se actualizan de acuerdo con cada componente de su nueva velocidad. La figura 4.2 ilustra de manera general la actualización de una partícula en un espacio de búsqueda de dos dimensiones.

El rol de la distancia a la mejor posición personal en la ecuación (4.1) es influir en la dirección de la partícula para que esta vuelva hacia dicha mejor posición en la cual ha encontrado mayor provecho, luego de que por largo tiempo la partícula haya estado explorando zonas del espacio de búsqueda en la cual no ha hallado mejoría. A esta distancia se denomina *factor*

cognitivo, el cual representa un conocimiento previo que la partícula ha determinado en base a sus experiencias individuales [Kennedy95].

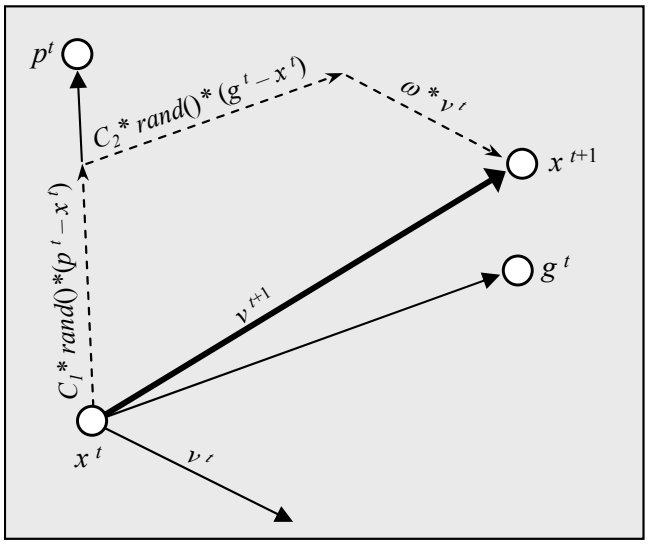


Figura 4.2. Ilustración de la actualización de una partícula en un espacio de búsqueda de dos dimensiones.

Por otra parte, el papel de la componente que representa la distancia a la mejor posición global es dirigir a las partículas hacia la posición más favorable hallada por el grupo. Tal distancia se denomina *factor social*, debido a que se trata de una información compartida dentro del enjambre de partículas que representa un modelo generalmente aceptado a seguir [Kennedy95].

Los parámetros C_1 y C_2 llamados *parámetro cognitivo* y *parámetro social* respectivamente, son constantes de aceleración hacia las mejores posiciones encontradas [Kennedy95], cuyos valores determinan la influencia máxima de las distancias a estas mejores posiciones en el cálculo de la nueva velocidad de las partículas. Los valores aleatorios retornados en las distintas llamadas a la función *rand*() son utilizados para determinar la influencia real de las distancias a las mejores posiciones, cuyo efecto aleatorio ayuda a preservar la diversidad en la población.

El parámetro de inercia ω, introducido por Eberhart y Shi [Shi98a], es utilizado para controlar el impacto de las velocidades anteriores en el cálculo de la nueva velocidad de las partículas. Este parámetro controla el balance entre la exploración global y local del espacio de búsqueda [Shi98b]. Un valor grande para este parámetro permite que se exploren nuevas áreas del espacio de búsqueda (*exploración*), mientras que un valor pequeño fomenta la exploración del área cercana a la posición actual de la partícula (*explotación*). Un buen valor para el parámetro de inercia es aquel que logra un balance apropiado entre la exploración de nuevas áreas y la explotación del área donde se encuentra la partícula [Shi98b]. Por otra parte, cuando el

valor de este parámetro es menor a 1 evita que las velocidades de las partículas crezcan sin control durante el transcurso de las iteraciones [Vesterstrom02].

Otra alternativa para evitar que las velocidades de las partículas crezcan desmesuradamente y controlar sus habilidades de exploración, consiste en limitar el nuevo valor de las componentes del vector de velocidad de las partículas a un valor máximo v_{max} [Shi98b], lo cual se realiza luego del cálculo de cada componente de las nuevas velocidades según:

$$\text{Si } v_{ij} > v_{max} \text{ entonces } v_{ij} = v_{max}$$

$$\text{Sino si } v_{ij} < -v_{max} \text{ entonces } v_{ij} = -v_{max}$$

En problemas de optimización que posean uno o más óptimos locales, la influencia de la mejor posición global en todas las partículas podría llevar a un caso de convergencia prematura hacia un óptimo local. Para ello, existe una variante de la metaheurística PSO [Eberhart95] en la cual se asocia cada partícula con un subgrupo de partículas del enjambre (en la que también se encuentra incluida la partícula) al que se denomina *vecindad de la partícula*. De esta forma, cada partícula elige a la mejor de entre las mejores posiciones personales en su vecindad para ser utilizada en lugar de la mejor posición global en el cálculo de su nueva velocidad, la cual se denota por $L_i = [l_{i1}, l_{i2},\ldots, l_{iN}]$ y se denomina *mejor posición local*. Así, con el uso de vecindades la ecuación (4.1) se convierte en:

$$v_{ij}^{t+1} = \omega \times v_{ij}^{t} + C_1 \times rand(\,) \times \left(p_{ij}^{t} - x_{ij}^{t}\right) + C_2 \times rand(\,) \times \left(l_{ij}^{t} - x_{ij}^{t}\right). \tag{4.3}$$

La vecindad de cada partícula puede ser establecida de la manera geográfica o lógica. Una vecindad geográfica consiste en establecer como vecinas de una partícula dada a todas las partículas que se encuentren separadas hasta cierta distancia máxima de ella o establecer como vecinas a todas las partículas que estén ubicadas dentro de cierta región del espacio de búsqueda. Una vecindad lógica se define estableciendo relaciones de asociación entre ciertas partículas del enjambre independientemente de la posición geográfica de cada una de ellas.

4.3 Optimización de Enjambre de Partículas en Problemas de Optimización Mono-objetivos

En la figura 4.3 puede observarse el algoritmo PSO para realizar una optimización mono-objetivo que utiliza la información de la mejor posición global para el cálculo de las nuevas velocidades de las partículas.

En las líneas 1 a 6 se establecen los valores para los parámetros del algoritmo. En las líneas 7 a 23 las partículas del enjambre son inicializadas con una posición y velocidad determinadas de manera aleatoria, la mejor posición personal de cada partícula es inicializada a su

posición inicial y la mejor de todas ellas es señalada como mejor posición global mediante el índice *g*.

```
1.  C1 := 2;    /* Parámetro cognitivo      */
2.  C2 := 2;    /* Parámetro social         */
3.  ω  := 0.8;  /* Parámetro de inercia     */
4.  vmax := 4;  /* Velocidad máxima         */
5.  g  := 1;    /* Índice del mejor global  */
6.  nro_partículas := 20; /* Número de partículas     */
7.  FOR i := 1 TO nro_partículas DO
8.     /* Inicializar la posición y velocidad de las partículas */
9.     FOR j := 1 TO N DO  /* Para cada componente */
10.       part[i].pos[j] := random_pos();
11.       part[i].vel[j] := random_vel();
12.    END_FOR
13.    part[i].eval := evaluar(part[i].pos); /* según función objetivo */
14.    /* Inicializar mejor posición personal a la posición actual */
15.    FOR j := 1 TO N DO  /* Para cada componente */
16.       mejor[i].pos[j] := part[i].pos[j];
17.    END_FOR
18.    mejor[i].eval := part[i].eval;
19.    /* Determinar la mejor posición global */
20.    IF(mejor[i].eval ES_MEJOR_QUE mejor[g].eval) THEN
21.       g := i;
22.    END_IF
23. END_FOR
24. WHILE(NOT condición_de_parada()) DO /* Realizar las iteraciones  */
25.    FOR i := 1 TO nro_partículas DO /* Actualizar las partículas  */
26.       FOR j := 1 TO N DO /* Para cada componente */
27.          part[i].vel[j] := ω * part [i].vel[j] +
                 C1 * rand() * (mejor[i].pos[j] – part[i].pos[j]) +
                 C2 * rand() * (mejor[g].pos[j] – part[i].pos[j]);
28.          IF(part[i].vel[j] > vmax) THEN
29.             part[i].vel[j] := vmax;
30.          ELSE IF(part[i].vel[j] < -vmax) THEN
31.             part[i].vel[j] := -vmax;
32.          END_IF
33.          part[i].pos[j] := part[i].pos[j] + part[i].vel[j];
34.       END_FOR
35.    END_FOR
36.    FOR i := 1 TO nro_partículas DO  /* Evaluar las partículas */
37.       /* Evaluar la posición actual de la partícula */
38.       part[i].eval := evaluar(part[i].pos); /* según función objetivo */
39.       /* Guardar la mejor posición personal de la partícula */
40.       IF (part[i].eval ES_MEJOR_QUE mejor[i].eval) THEN
41.          FOR j := 1 TO N DO  /* Para cada componente */
42.             mejor[i].pos[j] := part[i].pos[j];
43.          END_FOR
44.          mejor[i].eval := part[i].eval;
45.          /* Determinar la mejor posición global */
46.          IF(mejor[i].eval ES_MEJOR_QUE mejor[g].eval) THEN
47.             g := i;
48.          END_IF
49.       END_IF
50.    END_FOR
51. END_WHILE
52. retornar(mejor[g].pos[]); /* retornar mejor solución encontrada */
```

Figura 4.3. Algoritmo PSO para realizar una optimización mono-objetivo.

En las líneas 24 a 51 se encuentra el código correspondiente a las iteraciones del algoritmo PSO, las cuales se realizan hasta que se cumpla con alguna condición de parada tal como un número máximo de iteraciones, un tiempo máximo de ejecución o se halle una solución deseada. En las líneas 25 a 35, la velocidad y posición de cada partícula son actualizadas de

acuerdo con las ecuaciones (4.1) y (4.2), manteniendo los valores de las componentes de los vectores de velocidad dentro del intervalo [$-v_{max}$, v_{max}].

La evaluación de la nueva posición de cada partícula y la actualización de las mejores posiciones se realiza entre las líneas 36 a 50. En la línea 38 se procede a la evaluación de la posición de cada partícula a través de la función *evaluar*(), la cual implementa la función de evaluación propia del problema que se pretende resolver y retorna un valor escalar que indica el *valor de adaptabilidad* para la posición actual de la partícula. Luego en las líneas 40 a 49, la posición de cada partícula es comparada con su mejor posición personal y con la mejor posición global a través de sus valores de adaptabilidad. Si se determina que la posición actual de la partícula es una *mejor solución* que su mejor posición personal, esta última es actualizada haciéndola igual a la primera. De la misma manera, si la mejor posición personal de la partícula es una *mejor solución* que la mejor posición global indicada por el índice *g*, este índice es actualizado de manera que señale a esta mejor posición personal como mejor posición global. El algoritmo finaliza en la línea 52 retornando la mejor posición encontrada, la cual índica la mejor solución hallada por el enjambre.

En la figura 4.4 se ilustra la evolución de la evaluación de la mejor posición global con respecto al número de iteración al aplicar el algoritmo PSO para resolver la siguiente ecuación de igualdad:

$$x_1^2 + x_2^2 = 0\,, \tag{4.4}$$

donde $\vec{x} = [x_1, x_2]$ constituye la variable de decisión del problema.

La función de evaluación utilizada devuelve como valor de adaptabilidad el resultado de la evaluación del lado izquierdo de la ecuación (4.4) para la posición de una partícula. Para dos valores de adaptabilidad se considera mejor al de menor valor.

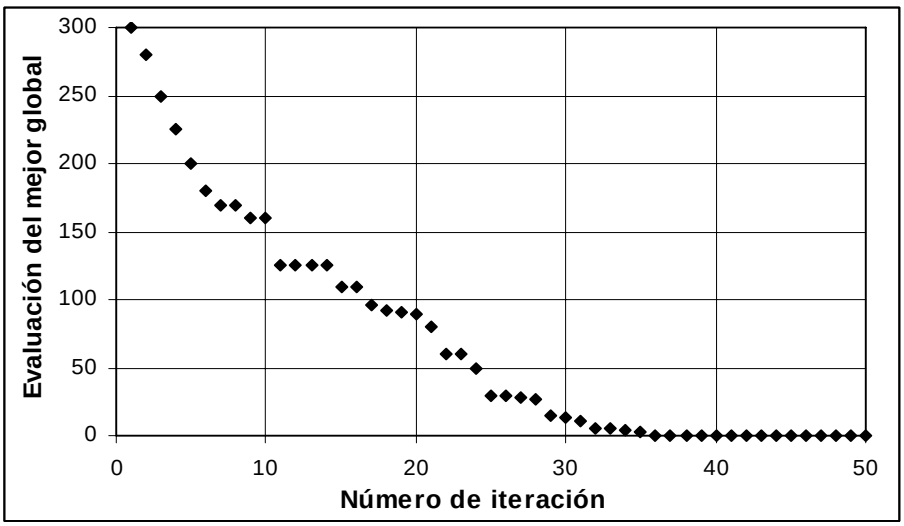


Figura 4.4. Evolución de la evaluación de la mejor posición global con respecto al número de iteración.

4.4 Efectos de los Parámetros del Algoritmo PSO

En el algoritmo de PSO existen varios parámetros (C_1, C_2, ω y v_{max}) que influyen en la manera en que las partículas exploran el espacio de búsqueda de soluciones. Debido a esto, es necesario tener una idea de como establecer los valores para cada uno de estos parámetros de manera a obtener resultados óptimos al utilizar la metaheurística PSO.

Un establecimiento inicial efectivo de estos parámetros para la resolución de un problema de optimización continuo a través de la metaheurística PSO consiste en hacer los valores de C_1, C_2 y ω iguales a 2, 2, y 0.8 respectivamente, y v_{max} igual al 10% del tamaño de espacio de búsqueda [Vesterstrom02].

A continuación se intenta dar una idea de cómo influye cada uno de estos parámetros en el movimiento de las partículas a través del espacio de búsqueda de soluciones.

4.4.1 Los Parámetros Social y Cognitivo

Los parámetros cognitivo (C_1) y social (C_2) influyen en la determinación de la dirección que seguirán las partículas hacia las mejores posiciones encontradas.

Cuando $C_2 >> C_1$ (léase C_2 es mucho mayor que C_1), las partículas son más atraídas por la mejor solución encontrada por el enjambre o dentro de su vecindad que por sus mejores posiciones personales. En el caso en que $C_1 = 0$ las partículas son sólo atraídas por la mejor posición determinada de manera grupal. En ambos casos todas las partículas intentan mejorar aun más la mejor solución global o local, lo cual puede ocasionar que la resolución del problema de optimización quede atascada en uno o más óptimos locales [Vesterstrom02].

En el otro caso, cuando $C_1 >> C_2$ todas las partículas intentan mejorar sus mejores posiciones personales en detrimento de la mejor posición global o local. En el límite, cuando $C_2 = 0$ cada partícula solo intenta mejorar su propia mejor posición. Bajo estas circunstancias, se desaprovecha el efecto acelerador que la información proveniente de las experiencias de las demás partículas puede tener en la búsqueda de soluciones, debido a que las partículas son más propensas a quedar explorando la zona alrededor de sus mejores soluciones personales.

Cuando $C_1 = C_2$ las partículas son atraídas en la misma proporción hacia las mejores soluciones encontradas, razón por la cual este suele ser el caso usualmente aplicado.

Otro punto a considerar tiene relación a la magnitud de los parámetros C_1 y C_2. Cuando estos tienen una magnitud alta las partículas reaccionan más rápidamente ante los cambios en las mejores posiciones, mientras que cuando presentan una magnitud baja ocurre lo contrario. Un establecimiento ideal para la magnitud de estos parámetros es aquel que permita a las par-

tículas reaccionar rápidamente ante los cambios en las mejores posiciones pero sin ocasionar que ellas abandonen precipitadamente el área que se encuentran explorando.

4.4.2 El Parámetro de Inercia

El parámetro de inercia ω permite controlar que tanto de la dirección actual de la partícula influye en el cálculo de su nueva velocidad [Shi98b].

Cuando $\omega = 0$, la dirección actual de la partícula no influye en su nueva velocidad y la partícula solo tiende a moverse hacia las mejores posiciones actuales. Si $0 < \omega << 1$, solo una pequeña fracción de la velocidad actual de la partícula influye en su nueva velocidad. En ambos casos las partículas son más propensas a reaccionar rápidamente ante los cambios de las mejores posiciones.

Si $\omega > 1$ las partículas tienden a conservar mucho de su dirección actual, lo que provoca que éstas presenten muy lenta reacción ante los cambios en las mejores posiciones cuando estas se encuentran en sentido contrario a su dirección de movimiento y viceversa. El hecho de $\omega > 1$ también puede implicar un problema, puesto que provoca que las velocidades de las partículas crezcan sin control durante el transcurso de las iteraciones.

Como a través del parámetro de inercia es posible controlar que tanto la partícula se guía por las mejores posiciones encontradas o por su dirección actual [Shi98b], se dice que este parámetro determina el balance entre la búsqueda de soluciones en las áreas próximas a las mejores soluciones encontradas, a lo que se denomina *explotación*, y la búsqueda de soluciones en nuevas áreas del espacio de búsqueda, a lo que se denomina *exploración*.

4.4.3 El Parámetro de Velocidad Máxima

El parámetro de velocidad máxima v_{max} limita la magnitud que puedan llegar a tener las componentes de los vectores de velocidad de las partículas y de esta forma limita el traslado total de las partículas entre una iteración y otra.

Si la magnitud de este parámetro es muy pequeña puede provocar que la partícula se desplace muy lentamente, aumentando el tiempo necesario para localizar una solución óptima o para que las partículas se desplacen hacia las mejores soluciones encontradas[Vesterstrom02].

Por otra parte, si la magnitud es muy grande la influencia de este parámetro puede no ser notoria y en el caso de que las velocidades de las partículas crezcan muy rápidamente, solo limita el cambio de posición de ellas a grandes saltos en el espacio de decisión.

Por lo general este parámetro no es utilizado en la mayoría de las implementaciones del algoritmo PSO, pero puede ser útil incluirlo con un valor relativamente alto que posteriormente se vaya haciendo disminuir hasta encontrar empíricamente su valor apropiado para el problema a resolver.

4.4.4 La Topología de Vecindad Lógica

La topología de vecindad lógica se refiere a la manera de establecer las asociaciones sociales entre las partículas del enjambre para la determinación de la mejor posición local. Cuando se utiliza el algoritmo PSO que usa la mejor posición global para guiar a las partículas, se dice que existe una única vecindad en la que cada partícula es vecina de todas las demás (vecindad global).

Existen dos topologías lógicas comúnmente utilizadas [Vesterstrom02], aunque pueden ser definidas otras. La topología *k-nearest*, que consiste en la asociación de cada partícula con las k partículas más cercanas en el espacio de índice de las partículas, por ejemplo para $k = 2$, la i-ésima partícula tiene como vecinas a las partículas i–1 e i+1 (nótese que para la partícula 1 sus vecinas son las partículas N y 2, mientras que para la partícula N sus vecinas son las partículas N-1 y 1). La topología de rueda (*wheel*) consiste en el establecimiento de una de las partículas del enjambre como vecina de todas las demás partículas.

Estas vecindades son ilustradas en la figura 4.5 en donde las partículas directamente conectadas deben ser interpretadas como vecinas.

Cabe hacer notar que cuando se utiliza este esquema se definen N vecindades, una para cada partícula del enjambre y que además cada partícula está incluida dentro de su propia vecindad.

Como fuera mencionado anteriormente, la mejor de las mejores posiciones personales de las partículas en la vecindad de la i-ésima partícula es utilizada en lugar de la mejor posición global para el cálculo de la nueva velocidad de dicha partícula.

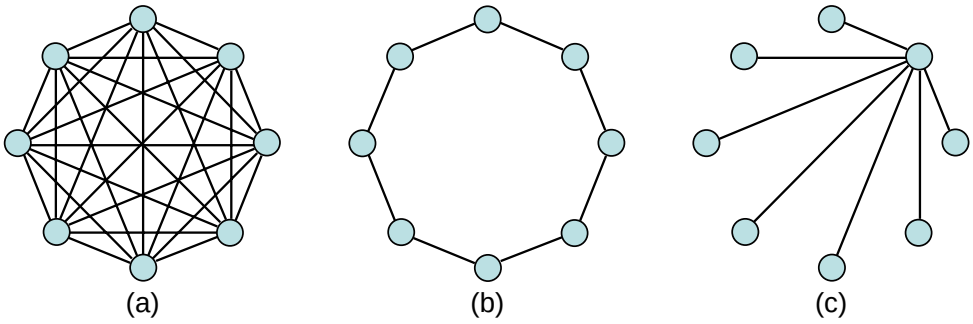


Figura 4.5. Esquema de vecindades: (a) topología totalmente conectada o de vecindad global, (b) topología k-nearest con k = 2 y (c) topología de rueda.

4.5 Optimización de Enjambre de Partículas en Problemas de Optimización Multiobjetivo

Adaptar el algoritmo PSO para realizar una optimización multiobjetivo requiere que en el cálculo de la nueva velocidad de las partículas (ecuación (4.1)) la mejor posición global sea una solución no dominada encontrada por el grupo y que la mejor posición personal sea una solución no dominada encontrada por la partícula. Para ello se utiliza un repositorio global que almacena las soluciones no dominadas entre si encontradas por todas las partículas y un repositorio personal para cada partícula que guarda las soluciones no dominadas entre si encontradas por esta.

En la Figura 4.6 se muestra el algoritmo genérico PSO para realizar una optimización multiobjetivo. En las líneas 1 a 5 se establecen los valores para los parámetros del algoritmo. La inicialización de las partículas y de los repositorios de soluciones no dominadas ocurre entre las líneas 6 y 22. En las líneas 8 a 11 cada partícula es inicializada con una posición y velocidad aleatoria. Luego, en la línea 12 la posición inicial de cada partícula es evaluada según la función objetivo vectorial del problema, cuyo resultado es almacenado en su vector de evaluación. En las líneas 14 a 17 se realiza la actualización del repositorio global de soluciones no dominadas, insertando las soluciones (posiciones en el espacio de búsqueda y en el espacio objetivo) no dominadas por los elementos contenidos en el repositorio y eliminando las soluciones dominadas por los nuevos elementos. De igual manera, en las líneas 18 a 21 se procede a la actualización del repositorio personal de cada partícula.

Entre las líneas 23 y 49 se realizan las iteraciones del algoritmo hasta que se cumpla alguna condición de parada tales como un número máximo de iteraciones o un tiempo máximo de ejecución. En las líneas 24 a 36 se actualizan la velocidad y la posición de cada partícula, para ello en la línea 25 se selecciona de los repositorios de soluciones no dominadas (global y/o personales) o de la población de partículas a una solución que haga las veces de mejor posición global y en la línea 26 se selecciona una solución del repositorio personal de soluciones no dominadas de la partícula como mejor posición personal. En las líneas 27 a 35 se realizan los cálculos de la nueva velocidad y posición de la partícula según las ecuaciones (4.1) y (4.2). En las líneas 39 a 47 se evalúan las nuevas posiciones de las partículas y se actualizan los repositorios. Finalmente, en la línea 52 se finaliza el algoritmo retornando todas las soluciones no dominadas almacenadas en el repositorio global.

```
1.  C1 := 2;         /* Parámetro cognitivo     */
2.  C2 := 2;         /* Parámetro social        */
3.  ω  := 0.8;       /* Parámetro de inercia    */
4.  vmax := 4        /* Velocidad máxima        */
5.  nro_partículas := 20; /* Número de partículas */
6.  FOR i := 1 TO nro_partículas DO /* inicializar las partículas */
7.     /* Inicializar la posición y velocidad de la partícula */
8.     FOR j := 1 TO N DO /* Para cada componente */
9.        part[i].pos[j] := random_pos();
10.       part[i].vel[j] := random_vel();
11.    END_FOR
12.    part[i].eval[] := evaluar(part[i].pos[]); /* según ec. (2.1) */
13.    /* Inicializar los repositorios */
14.    IF (part[i] NO_ES_DOMINADA_POR P, ∀P ∈ rep_global) THEN
15.       Incluir_partícula(rep_global, part[i]);
16.       Eliminar_dominados(rep_global);
17.    END_IF
18.    IF (part[i] NO_ES_DOMINADA_POR P, ∀P ∈ rep_personal[i]) THEN
19.       Incluir_partícula(rep_personal[i], part[i]);
20.       Eliminar_dominados(rep_personal[i]);
21.    END_IF
22. END_FOR
23. WHILE(NOT condición_de_parada()) DO /* Realizar las iteraciones  */
24.    FOR i := 1 TO nro_partículas DO /* Actualizar las partículas */
25.       mejor_global := Seleccionar_partícula(rep_global, rep_personal[],
                          part[], i);
26.       mejor_personal := Seleccionar_partícula(rep_personal[i]);
27.       FOR j := 1 TO N DO /* Para cada componente */
28.          part[i].vel[j] := ω * part [i].vel[j] +
                c1 * rand() * (mejor_global.pos[j] – part[i].pos[j]) +
                c2 * rand() * (mejor_personal.pos[j]  – part[i].pos[j]);
29.          IF(part[i].vel[j] > vmax) THEN
30.             part[i].vel[j] := vmax
31.          ELSE IF(part[i].vel[j] < -vmax) THEN
32.             part[i].vel[j] := -vmax;
33.          END_IF
34.          part[i].pos[j] := part[i].pos[j] + part[i].vel[j];
35.       END_FOR
36.    END_FOR
37.    FOR i := 1 TO nro_partículas DO  /* Evaluar las partículas */
38.       part[i].eval[] := evaluar(part[i].pos);
39.       /* Actualizar los repositorios */
40.       IF (part[i] NO_ES_DOMINADA_POR P, ∀P ∈ rep_global) THEN
41.          Incluir_partícula(rep_global, part[i]);
42.          Eliminar_dominados(rep_global);
43.       END_IF
44.       IF (part[i] NO_ES_DOMINADA_POR P, ∀P ∈ rep_personal[i]) THEN
45.          Incluir_partícula(rep_personal[i], part[i]);
46.          Eliminar_dominados(rep_personal[i]);
47.       END_IF
48.    END_FOR
49. END_WHILE
50. Retornar_soluciones_no_dominadas(rep_global);
```

Figura 4.6. Algoritmo genérico PSO para realizar una optimización multiobjetivo.

Básicamente, los distintos métodos que tratan la aplicación de la metaheurística PSO a problemas multiobjetivos se diferencian entre si en la manera en como eligen las mejores posiciones de los repositorios para realizar la actualización de las partículas.

A continuación se presentan algunos métodos que tratan la aplicación de la metaheurística PSO para resolver MOPs, encontrados en la literatura y denominados métodos MOPSO (*MultiObjective Particle Swarm Optimization*).

4.5.1 Moore y Chapman (1999)

Moore y Chapman [Moore99] proponen un método MOPSO en el cual cada partícula almacena en un repositorio personal todas las soluciones no dominadas entre si que ha encontrado durante la exploración.

Para realizar el cálculo de su nueva velocidad, cada partícula toma como mejor posición personal a cualquier elemento de su repositorio personal y como mejor posición global a cualquier elemento de su repositorio personal no dominado por los elementos en los repositorios personales de las partículas de su vecindad.

Moore y Chapman [Moore99] no mencionan que debe realizarse en el caso de que todas las soluciones en el repositorio personal de la partícula sean dominadas por las soluciones en los repositorios personales de las partículas de su vecindad. Bajo esta situación, la implementación de este MOPSO realizada para este trabajo elige como mejor global a cualquier elemento de su repositorio personal, de manera a mantenerla lo más parecida a la propuesta por sus autores. Además, la implementación incluye un repositorio global externo en cual se almacenan todas las soluciones no dominadas encontradas durante la ejecución del algoritmo. El pseudocódigo para este método MOPSO basado en el algoritmo genérico de la figura 4.6 se presenta en la figura a continuación.

```
1.     Inicializar los parámetros c1, c2, ω y vmax;
2.  FOR i := 1 TO nro_partículas DO
3.     Inicializar la i-ésima partícula con una posición y velocidad aleatoria;
4.     Evaluar la posición de la i-ésima partícula según ec. (2.1);
5.     Actualizar el repositorio global;
6.     Actualizar el repositorio personal de la i-ésima partícula;
7.  END_FOR
8.  WHILE(NOT condición_de_parada()) DO
9.     FOR i := 1 TO nro_partículas DO
10.       /* seleccionar mejor posición personal */
11.       mejor_personal := Seleccionar cualquier elemento del repositorio
                            personal de la i-ésima partícula;
12.       /* seleccionar mejor posición global */
13.       mejor_global := Seleccionar cualquier elemento del repositorio personal
                          de la i-ésima partícula no dominado por los elementos de
                          los repositorios de las partículas de su vecindad;
14.       Actualizar la velocidad de la i-ésima partícula según ec. (4.1);
15.       Limitar las componentes de la velocidad a vmax;
16.       Actualizar posición de la i-ésima partícula según ec. (4.2);
17.    END_FOR
18.    FOR i := 1 TO nro_partículas DO
19.       Evaluar la posición de la i-ésima partícula según ec. (2.1);
20.       Actualizar el repositorio global;
21.       Actualizar el repositorio personal de la i-ésima partícula;
22.    END_FOR
23. END_WHILE
24. Retornar_soluciones_no_dominadas(repositorio global);
```

Figura 4.7. Pseudocódigo del método MOPSO de Moore y Chapman.

4.5.2 Hu y Eberhart (2002)

Hu y Eberhart [Hu02] proponen un método MOPSO para problemas de optimización de dos objetivos, en el cual cada miembro de la población de partículas elige como mejor global a una de *m* soluciones previamente seleccionadas del repositorio global de soluciones no dominadas, donde *m* es un parámetro propio de este método que debe ser definido a priori e indica el número máximo de elementos a considerar en la selección.

El criterio de selección se basa en un objetivo específico, eligiéndose para cada partícula a las *m* soluciones del repositorio global más cercanas a ella en valor con respecto a la evaluación del objetivo específico. De las *m* soluciones elegidas, aquella solución con mejor evaluación en el objetivo restante es elegida como mejor global para la partícula en cuestión.

Cada partícula del enjambre mantiene en su repositorio personal solo una posición, la cual es remplazada cuando la partícula encuentra una nueva posición que la domina.

El pseudocódigo para este método MOPSO basado en el algoritmo de la figura 4.6 se muestra en la figura 4.8.

```
1.  Inicializar los parámetros c1, c2, ω y vmax;
2.  obj_de_proximidad := f2(x); /* objetivo de proximidad */
3.  obj_de_evaluación := f1(x); /* objetivo de evaluación */
4.  m := 2; /* número máximo de soluciones a considerar */
5.  FOR i := 1 TO nro_de_partículas DO
6.     Inicializar la i-ésima partícula a una posición y velocidad aleatoria;
7.     Evaluar la i-ésima partícula según ec. (2.1);
8.     Actualizar el repositorio global;
9.     Actualizar el repositorio personal de la i-ésima partícula;
10. END_FOR
11. WHILE(NOT condición_de_parada()) DO
12.    FOR i := 1 TO nro_de_partículas DO
13.       /* seleccionar mejor posición personal */
14.       mejor_personal := posición almacenada en el repositorio personal;
15.       /* seleccionar mejor posición global */
16.       Seleccionar del repositorio global a las m partículas más cercanas
          en valor para la i-ésima partícula con respecto a obj_de_proximidad;
17.       mejor_global := Elegir a la partícula con mejor evaluación en
                          obj_de_evaluación de entre las m partículas
                          seleccionadas;
18.       Actualizar la velocidad de la i-ésima partícula según la ec. (4.1);
19.       Limitar las componentes de la velocidad a vmax;
20.       Actualizar posición de la i-ésima partícula según la ec. (4.2);
21.    END_FOR
22.    FOR i := 1 TO nro_de_partículas DO
23.       Evaluar la i-ésima partícula según ec. (2.1);
24.       Actualizar el repositorio global;
25.       Actualizar el repositorio personal de la i-ésima partícula;
26.    END_FOR
27. END_WHILE
28. Retornar_soluciones_no_dominadas(repositorio global);
```

Figura 4.8. Pseudocódigo del método MOPSO de Hu y Eberhart.

4.5.3 Coello y Lechuga (2002)

Coello y Lechuga [Coello02] presentan un método MOPSO en la cual se mantiene un repositorio global de tamaño fijo que almacena las soluciones no dominadas encontradas por todas las partículas durante el proceso de búsqueda.

Este repositorio utiliza un esquema de *malla adaptativa* similar al empleado por el método evolutivo PAES [Knowles00], en el cual el espacio objetivo explorado se representa mediante regiones acotadas llamadas *hipercubos*. La denominación de *adaptativa* proviene de la capacidad de los hipercubos de auto ajustar sus límites cuando se intenta insertar una solución que no cae dentro de un hipercubo.

Cada hipercubo recibe un valor de calificación igual al resultado de dividir el número 10 entre el número de soluciones no dominadas situadas dentro de él. Los hipercubos que no poseen soluciones no dominadas situadas dentro de ellos reciben un valor de calificación igual a 0. Esta calificación es posteriormente utilizada para la selección de un elemento del repositorio a ser utilizado como mejor posición global para cada partícula del enjambre. La figura 4.9 ilustra un ejemplo de la división en hipercubos de un espacio objetivo bidimensional.

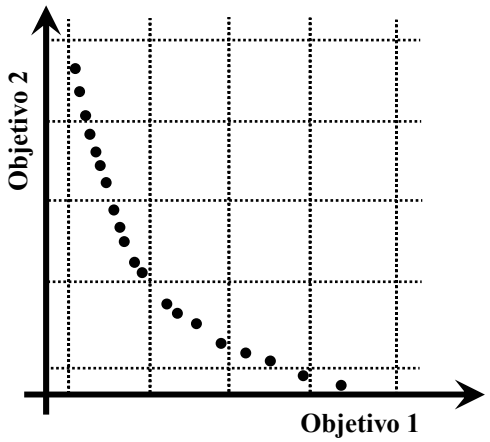


Figura 4.9. Ilustración de la división en hipercubos.

El repositorio global es actualizado tras cada iteración insertando todas las soluciones de la población actual que no sean dominadas por los elementos existentes y eliminando los elementos que resulten dominados. Si se debe insertar una solución dentro del repositorio global cuando está lleno, se elimina una solución del hipercubo más poblado y se procede a insertar la nueva solución.

El algoritmo de asignación de los elementos del repositorio global a los hipercubos es ilustrado en la figura 4.10. Este algoritmo es corrido luego de la inserción de los elementos no dominados en la primera generación de partículas y cuando un elemento del repositorio no cae dentro de algún hipercubo.

En la figura 4.11 se muestra el algoritmo para determinar el hipercubo correspondiente a una solución recientemente insertada dentro del repositorio, si en este algoritmo se determina que no existe un hipercubo para almacenar una solución se llama al algoritmo de la figura 4.10 para readaptar los hipercubos.

```
1.  Recibir el repositorio de soluciones no dominadas R;
2.  Recibir el número de divisiones en cada objetivo nro_div;
3.  nro_hipercubos := pow(nro_div, nro_objetivos); // número de hipercubos
4.  Hipercubo := Inicializar_hipercubos(nro_hipercubos); // creación de los hipercubos
5.  FOR i := 1 TO nro_objetivos DO // determinar los valores extremos
6.     max[i] = R[1].eval[i];  // valor máximo en cada objetivo
7.     min[i] = R[1].eval[i];  // valor mínimo en cada objetivo
8.     FOR j:= 2 TO |R| DO
9.        IF(max[i] < R[j].eval[i]) THEN
10.          max[i] := R[j].eval[i];
11.       ELSE IF(min[i] > R[j].eval[i]) THEN
12.          min[i] := R[j].eval[i];
13.       END_IF
14.    END_FOR
15. END_FOR
16. FOR i := 1 TO nro_objetivos DO // determinar dimensiones de los hipercubos
17.    // amplitud de los hipercubos en cada objetivo
18.    amp[i] := (max[i] – min[i]) / (nro_div - 1);
19.    // coordenadas de inicio en cada objetivo
20.    ini[i] := min[i] – amp[i] / 2;
21. END_FOR
22. FOR i:= 1 TO |R| DO // |R| representa la cardinalidad de R
23.    pos := 0; // número del hipercubo para la solución
24.    FOR j:= 1 TO nro_objetivos DO
25.       aux := floor((R[i].eval[j] - ini[j]) / amp[j]);
26.       pos := pos + aux * pow(nro_div, j);
27.    END_FOR
28.    insertar(Hipercubo[pos], R[i]); // insertar solución en hipercubo
29. END_FOR
30. FOR i:= 1 TO nro_hipercubos DO // determinar las calificaciones de los hipercubos
31.    IF (|Hipercubo[i]| > 0) THEN
32.       Hipercubo[i].calif := 10 / |Hipercubo[i]|;
33.    ELSE
34.       Hipercubo[i].calif := 0;
35.    END_IF
36. END_FOR
```

Figura 4.10. Algoritmo de asignación de las soluciones a los hipercubos.

```
1.  Recibir solución S;
2.  IF (Repositorio_lleno() == TRUE) THEN
3.     Eliminar una solución del hipercubo más poblado;
4.  END_IF
5.  pos := 0;
6.  FOR i:= 1 TO nro_objetivos DO
7.     max := ini[i] + amp[i] * nro_div;
8.     IF(S.eval[i] < ini[i] OR S.eval[i] > max) THEN
9.        // por debajo o encima de los hipercubos
10.       Readaptar hipercubos; // algoritmo de la figura 4.14
11.       Retornar;
12.    END_IF
13.    aux := floor((S.eval[i] - ini[i]) / amp[i]);
14.    pos := pos + aux * pow(nro_div, i);
15. END_FOR
16. insertar(Hipercubo[pos], S); // insertar solución en hipercubo
17. Hipercubo[pos].calif := 10 / |Hipercubo[pos]|;
```

Figura 4.11. Algoritmo para determinar el hipercubo correspondiente a una solución

La selección de un elemento del repositorio como mejor posición global para cada partícula se realiza seleccionando un hipercubo mediante el método de la ruleta a través de su valor de calificación y luego eligiendo al azar cualquier elemento dentro del hipercubo seleccionado.

Cada partícula del enjambre sólo mantiene una solución en su repositorio personal, la cual es reemplazada cada vez que la partícula encuentra una nueva solución no dominada por esta. Cuando la partícula encuentra una solución no comparable con su mejor posición personal actual elige al azar a una de ellas como nueva mejor posición personal.

En la figura 4.12 se presenta el pseudocódigo para este método MOPSO basado en el algoritmo de la figura 4.6.

```
1.     Inicializar los parámetros c1, c2, ω y vmax;
2.  FOR i := 1 TO nro_de_partículas DO
3.     Inicializar la partícula i con una posición y velocidad aleatoria;
4.     Evaluar la partícula i según ec. (2.1);
5.     Actualizar el repositorio global y los hipercubos;
6.     Actualizar el repositorio personal de la i-ésima partícula;
7.  END_FOR
8.  WHILE(NOT condición_de_parada()) DO
9.     FOR i := 1 TO nro_de_partículas DO
10.       /* seleccionar mejor posición personal */
11.       mejor_personal := Posición almacenada en el repositorio personal de
                            la i-ésima partícula;
12.       /* seleccionar mejor posición global */
13.       Elegir un hipercubo h de repositorio global mediante el método
          de la ruleta;
14.       mejor_global := Seleccionar cualquier posición en el hipercubo h;
15.       Actualizar la velocidad de la i-ésima partícula según la ec. (4.1);
16.       Limitar las componentes de la velocidad a vmax;
17.       Actualizar posición de la i-ésima partícula según ec. (4.2);
18.    END_FOR
19.    FOR i := 1 TO nro_de_partículas DO
20.       Evaluar la partícula i según ec. (2.1);
21.       Actualizar el repositorio global y los hipercubos;
22.       Actualizar el repositorio personal;
23.    END_FOR
24. END_WHILE
25. Retornar las soluciones no dominadas en el repositorio global;
```

Figura 4.12. Pseudocódigo del método MOPSO de Coello y Lechuga.

4.5.4 Fieldsend y Singh (2002)

Fieldsend y Singh [Fieldsend02] proponen un método MOPSO en el cual se utiliza un repositorio global que almacena todas las soluciones no dominadas encontradas por las partículas y una estructura de datos llamada *dominated tree* [Fieldsend01], construida a partir de las soluciones en el repositorio global y a través de la cual cada partícula del enjambre elige a un elemento del repositorio global a ser empleado como mejor posición global en el cálculo de su nueva velocidad.

El *dominated tree* [Fieldsend01] consiste en una lista de L vectores c_i de dimensión M denominados *puntos compuestos*, cuyos valores para sus componentes son determinados a partir

de los elementos del repositorio global de soluciones no dominadas. El número de puntos compuestos L está dado por:

$$L = \lceil |R| / M \rceil, \tag{4.5}$$

donde R es el repositorio global de soluciones no dominadas, $|\cdot|$ representa la cardinalidad y M es el número de objetivos del problema a resolver.

En el *dominated tree* [Fieldsend01], los puntos compuestos C_i se encuentran ordenados por la relación de dominancia débil:

$$C_L \succeq C_{L\text{-}1} \succeq \ldots \succeq C_2 \succeq C_1$$

donde $C_i = \left[c_{i1}, c_{i2}, \ldots, c_{iM}\right] \in \Re^M \quad \forall i \in \{1 \ldots L\}$.

Cada punto compuesto C_i se asocia con M elementos del repositorio global R a partir de los cuales se determinan los valores para sus coordenadas. Así para el primer punto compuesto C_1, el valor de su primera coordenada es igual al valor de la primera coordenada del vector de evaluación de la solución x_m en R con mayor valor en dicha coordenada, lo que se denota por:

$$c_{1,1} = x_{m,1} \mid x_m \in R \wedge \nexists x_n \in R \Rightarrow x_{n,1} > x_{m,1},$$

donde $c_{1,1}$ es la primera coordenada del punto compuesto C_1, $x_{m,1}$ es la primera coordenada del vector de evaluación de la solución x_m y $x_{n,1}$ es la primera coordenada del vector de evaluación de la solución x_n. La solución x_m es asociada con el punto compuesto C_1, lo que se denota como $x_m \rightarrow C_1$, y ya no es considerada para la determinación del valor de otra coordenada.

De forma análoga, la segunda coordenada $c_{1,2}$ del punto compuesto C_1 se hace igual al valor de la segunda coordenada del vector de evaluación de la solución x_k en R con mayor valor en dicha coordenada y que no está asociada a algún punto compuesto, lo que se expresa como:

$$c_{1,2} = x_{k,2} \mid x_k \in R \wedge \nexists x_n \in R \Rightarrow x_{n,2} > x_{k,2} \wedge x_k \not\rightarrow C_i \quad \forall i \in \left[1 \ldots L\right].$$

De manera general, la determinación de la j-ésima componente del i-ésimo punto compuesto C_i se expresa como:

$$c_{i,j} = x_{m,j} \mid x_m \in R \wedge \nexists x_n \in R \Rightarrow x_{n,j} > x_{m,j} \wedge x_m \not\rightarrow C_i \quad \forall i \in \left[1 \ldots L\right].$$

La determinación de cada punto compuesto concluye una vez determinada todas sus componentes, quedando cada punto compuesto asociado a M elementos del repositorio de global de soluciones no dominadas.

El procedimiento para determinar todos los puntos compuestos se repite hasta asociar todos los elementos de R. Nótese que para la construcción del último punto compuesto C_L puede ocurrir que no resten suficientes elementos en R. Entonces, las coordenadas en C_L para las

cuales ya no existen elementos en R se hacen iguales a los valores de las coordenadas correspondientes al último elemento asociado a C_L. En la Figura 4.13 se muestra un ejemplo de un *dominated tree* en un espacio objetivo de dos dimensiones.

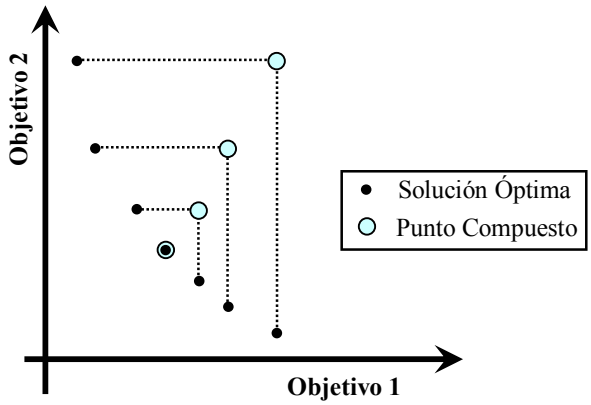


Figura 4.13. Ejemplo de un *dominated tree* en un espacio de dos dimensiones.

Debido a que el repositorio de soluciones óptimas R es constantemente actualizado durante la ejecución de este MOPSO, el *dominated tree* soporta las operaciones de inserción y borrado de soluciones, las cuales ayudan a mantener la correspondencia entre esta estructura de datos y el repositorio global de soluciones no dominadas. Los pseudocódigos para estas operaciones pueden ser encontrados en [Fieldsend01].

La elección del mejor global para un individuo p se realiza a través del *dominated tree* [Fieldsend01], buscando el primer punto compuesto C_i no dominado por la evaluación de la partícula p y luego eligiendo como mejor global a cualquier solución x asociada con el punto compuesto C_i que contribuya con un valor de coordenada que sea menor o igual al valor de la correspondiente coordenada del vector de evaluación de p. En la figura 4.14 se ilustra esta elección.

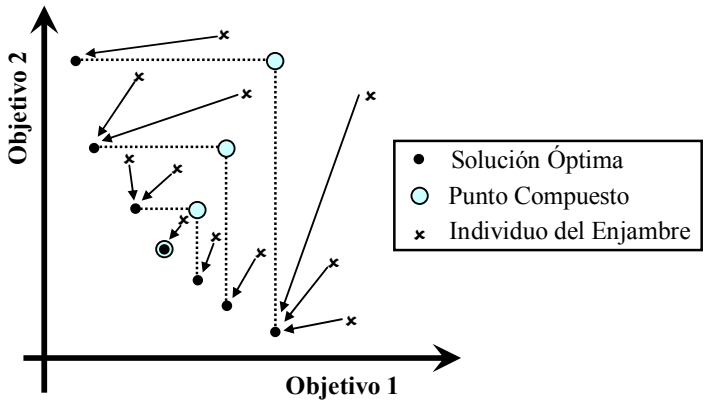


Figura 4.14. Elección del mejor global utilizando un *dominated tree*.

En el método MOPSO de Fieldsend y Singh [Fieldsend02], cada partícula mantiene en su repositorio personal todas las soluciones no dominadas entre si que ella ha encontrado. Para el

cálculo de su nueva velocidad, cada partícula elige al azar a una de las soluciones en su repositorio personal como mejor posición personal.

Este método MOPSO [Fieldsend02] también introduce un operador llamado turbulencia, el cual básicamente actúa como operador de mutación sobre la nueva posición de cada partícula. El operador de turbulencia es aplicado en la ecuación (4.2) de la siguiente manera:

$$x_{ij}^{t+1} = x_{ij}^{t} + v_{ij}^{t+1} + t\,, \tag{4.6}$$

donde t es una valor aleatorio en el intervalo [$-t_{MAX}$, t_{MAX}] y t_{MAX} es un parámetro propio de este método, definido a priori .

El pseudocódigo para el método MOPSO de Fielsend y Singh [Fieldsend02] basado en algoritmo de la figura 4.6 es presentado en la figura 4.15.

```
1.  Inicializar los parámetros c1, c2, ω y vmax;
2.  FOR i := 1 TO nro_de_partículas DO
3.     Inicializar la i-ésima partícula a una posición y velocidad aleatoria;
4.     Evaluar la i-ésima partícula según ec. (2.1);
5.     Actualizar el repositorio global y el dominated tree;
6.     Actualizar el repositorio personal del i-ésima partícula;
7.  END_FOR
8.  WHILE(NOT condición_de_parada()) DO
9.     FOR i := 1 TO nro_de_partículas DO
10.       /* seleccionar mejor posición personal */
11.       mejor_personal := Seleccionar cualquier posición almacenada en el
                            repositorio personal de la i-ésima partícula;
12.       /* seleccionar mejor posición global */
13.       mejor_global := Seleccionar una posición del repositorio global
                          a través del dominated tree;
14.       Actualizar la velocidad de la i-ésima partícula según la ec. (4.1);
15.       Limitar las componentes de la velocidad a vmax;
16.       Actualizar la posición de la i-ésima partícula según la ec. (4.6);
17.    END_FOR
18.    FOR i := 1 TO nro_de_partículas DO
19.       Evaluar la i-ésima partícula según ec. (2.1);
20.       Actualizar el repositorio global y el dominated tree;
21.       Actualizar el repositorio personal de la i-ésima partícula;
22.    END_FOR
23. END_WHILE
24. Retornar las soluciones no dominadas en el repositorio global;
```

Figura 4.15. Pseudocódigo del método MOPSO de Fieldsend y Singh.

4.5.5 Mostaghim y Teich (2003)

Mostaghim y Teich [Mostaghim03] proponen un método MOPSO que hace uso de un repositorio global para almacenar las soluciones no dominadas encontradas por las partículas del enjambre y de una función llamada *función sigma*, a través de la cual se selecciona un elemento del repositorio global para cada partícula a ser utilizado como mejor posición global en el cálculo de su nueva velocidad.

La función sigma asocia a cada solución un vector $\vec{\sigma}$ cuya dimensión es igual al número de combinaciones posibles de M elementos tomados de a 2, donde M es el número de objetivos del problema a resolver. En esta función, cada componente del vector $\vec{\sigma}$ se calcula como

la diferencia de los cuadrados de 2 componentes del vector de evaluación de la solución, la cual luego es dividida por la suma de los cuadrados de todas las componentes de dicho vector. A continuación se presentan las ecuaciones de la función sigma para valores de *M* iguales a 2, 3 y 4:

$$sigma(x) = \frac{(f_1(x))^2 - (f_2(x))^2}{(f_1(x))^2 + (f_2(x))^2} \qquad \text{para } M = 2 \qquad (4.7)$$

$$sigma(x) = \begin{bmatrix} (f_1(x)^2 - f_2(x)^2) \\ (f_2(x)^2 - f_3(x)^2) \\ (f_3(x)^2 - f_1(x)^2) \end{bmatrix} / (f_1(x)^2 + f_2(x)^2 + f_3(x)^2) \qquad \text{para } M = 3 \qquad (4.8)$$

$$sigma(x) = \begin{bmatrix} (f_1(x)^2 - f_2(x)^2) \\ (f_1(x)^2 - f_3(x)^2) \\ (f_1(x)^2 - f_4(x)^2) \\ (f_2(x)^2 - f_3(x)^2) \\ (f_2(x)^2 - f_4(x)^2) \\ (f_3(x)^2 - f_4(x)^2) \end{bmatrix} / (f_1(x)^2 + f_2(x)^2 + f_3(x)^2 + f_4(x)^2) \qquad \text{para } M = 4 \qquad (4.9)$$

La función *sigma* tiene la característica de asociar el mismo vector $\vec{\sigma}$ a todas las soluciones cuyas evaluaciones se encuentren en la misma línea hacia el origen de coordenadas en el espacio objetivo [Mostaghim03].

El repositorio global es utilizado para almacenar todas las soluciones no dominadas encontradas por las partículas del enjambre. En cada iteración, las soluciones presentes en la población de partículas que no son dominadas por las soluciones en el repositorio global son insertadas y las soluciones que resulten dominadas son eliminadas. Luego de la actualización del repositorio, la función *sigma* es evaluada para cada nueva solución insertada de manera a obtener el vector $\vec{\sigma}$ correspondiente al nuevo elemento.

Al momento de calcular la nueva velocidad para cada partícula del enjambre, la función sigma es evaluada para la solución correspondiente a la partícula obteniéndose su vector $\vec{\sigma}$ correspondiente. Luego, la distancia euclidiana entre el vector $\vec{\sigma}$ de la partícula y los vectores $\vec{\sigma}$ de los elementos del repositorio es computada y la partícula toma como mejor posición global al elemento del repositorio para el cual la distancia euclidiana entre sus vectores $\vec{\sigma}$ sea la menor. De esta forma, cada partícula toma como mejor posición global a la solución del repositorio que se encuentre en la línea hacia el origen de coordenadas en el espacio objetivo más próxima a la línea en que ella se encuentra, lo cual se ilustra en la figura 4.16.

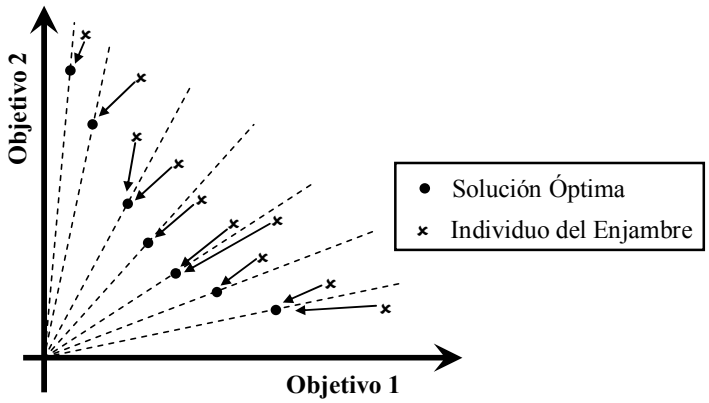


Figura 4.16. Selección de la mejor posición global mediante el uso de la función sigma.

Cada partícula de enjambre sólo mantiene en su repositorio personal una solución, la cual es reemplazada cada vez que la partícula encuentra una nueva solución que la domina.
El pseudocódigo para el método MOPSO de Mostaghim y Teich [Mostaghim03] basado en el algoritmo de la figura 4.6 se presenta en la figura 4.17.

```
1.  Inicializar los parámetros c1, c2, ω y vmax;
2.  FOR i := 1 TO nro_de_partículas DO
3.     Inicializar la i-ésima partícula con una posición y velocidad aleatoria;
4.     Evaluar la i-ésima partícula según ec. (2.1);
5.     Actualizar el repositorio global y calcular la función sigma
       para la nueva solución;
6.     Actualizar el repositorio personal de la i-ésima partícula;
7.  END_FOR
8.  WHILE(NOT condición_de_parada()) DO
9.     FOR i := 1 TO nro_de_partículas DO
10.       /* seleccionar mejor posición personal */
11.       mejor_personal := Posición almacenada en el repositorio personal
                            de la i-ésima partícula;
12.       /* seleccionar mejor posición global */
13.       Calcular la distancia entre el vector sigma de la solución indicada
          por la i-ésima partícula y los vectores sigma de las soluciones del
          repositorio global;
14.       mejor_global := Seleccionar la solución del repositorio global
                          para el cual la distancia entre los vectores sigma
                          sea la menor;
15.       Actualizar la velocidad de la i-ésima partícula según la ec. (4.1);
16.       Limitar las componentes de la velocidad a vmax;
17.       Actualizar posición de la i-ésima partícula según la ec. (4.2);
18.    END_FOR
19.    FOR i := 1 TO nro_de_partículas DO
20.       Evaluar la i-ésima partícula según ec. (2.1);
21.       Actualizar el repositorio global y calcular la función sigma para la
          nueva solución;
22.       Actualizar el repositorio personal de la i-ésima partícula;
23.    END_FOR
24. END_WHILE
25. Retornar las soluciones no dominadas en el repositorio global;
```

Figura 4.17. Pseudocódigo del método MOPSO de Mostaghim y Teich.

4.6 Adaptación del PSO para Resolver el TSP

Para las implementaciones de este trabajo se utilizó la adaptación del PSO para el TSP propuesta por Secrest [Secrest01]. Esta adaptación utiliza un algoritmo de construcción de solu-

ciones llamado PSO_AS para determinar las nuevas posiciones de las partículas en lugar de las ecuaciones (4.1) y (4.2) del algoritmo PSO, definiendo el espacio de búsqueda de soluciones como el conjunto de todas las permutaciones posibles de las N ciudades del problema TSP que se pretende resolver. De esta forma, la posición de una partícula corresponde a una permutación y las partículas recorren el espacio de búsqueda pasando de una permutación a otra.

El algoritmo de construcción de soluciones PSO_AS conserva la idea original del PSO de que la siguiente posición a visitar por una partícula se deduce a partir de su posición actual (X) y las mejores posiciones halladas por las partículas, más específicamente, la mejor posición del enjambre (G) y la mejor posición de la vecindad de la partícula (L). En este algoritmo, el factor cognitivo (C_1) y el factor social (C_2) son reemplazados por tres factores llamados *factores de influencia*, K_1 que indica la probabilidad de seleccionar información de la posición actual de la partícula, K_2 que indica la probabilidad de seleccionar información de la mejor posición de la vecindad de la partícula, y K_3 que indica la probabilidad de seleccionar información de la mejor posición del enjambre. Los valores para los factores K_1, K_2 y K_3 deben encontrase en el intervalo [0, 1] y su suma debe ser igual a 1.

El algoritmo de construcción de soluciones PSO_AS se ilustra en la figura 4.18. Primeramente en las líneas 1 a 5 se reciben la posición actual de la partícula (pos), la mejor posición encontrada hasta el momento dentro de la vecindad de la partícula (mejor_local), la mejor posición encontrada hasta ahora por el enjambre (mejor_global), los valores para los factores K_1, K_2 y K_3, y el número de ciudades del problema TSP. En la línea 7 se establece como la ciudad inicial para la nueva posición de la partícula (nueva_pos) a la última ciudad que aparece en la posición actual de la partícula.

Entre las líneas 8 y 27 se determinan de manera iterativa las siguientes N-2 ciudades para la nueva posición de la partícula. En la línea 9 se elige un número aleatorio en el intervalo [0, 1]. Si el valor del número aleatorio se encuentra en el intervalo [0, K_3] se elige como siguiente ciudad en la nueva posición de la partícula a la ciudad dentro de la mejor posición del enjambre que aparece luego de la última ciudad determinada para la nueva posición de la partícula, véase líneas 11 a 17. Si el valor del número aleatorio cae dentro del intervalo (K_3, K_3 + K_2] se elige como siguiente ciudad en la nueva posición de la partícula a la ciudad dentro de la mejor posición de la vecindad de la partícula que aparece luego de la última ciudad determinada para la nueva posición de la partícula, véase líneas 18 a 24. Si el número aleatorio está dentro del intervalo (K_3 + K_2, K_3 + K_2 + K_1] se elige como siguiente ciudad en la nueva posición de la partícula a la última ciudad dentro de la posición actual de la partícula que no aparece en la nueva posición, véase líneas 25 a 27.

Finalmente en la línea 29 se toma como última ciudad para la nueva posición de la partícula a la ciudad restante de las *N* ciudades del problema TSP y en la línea 30 se retorna la nueva posición para la partícula, la cual corresponde a una permutación de la *N* ciudades.

```
1.  Recibir pos_actual; /* posición actual de la partícula */
2.  Recibir mejor_local; /* mejor posición en la vecindad de la part. */
3.  Recibir mejor_global; /* mejor posición encontrada en el enjambre */
4.  Recibir los valores para los factores K1, K2 y K3;
5.  Recibir nro_ciudades; /* número de ciudades del problema TSP */
6.  /* nueva_pos indica la nueva posición para la partícula */
7.  nueva_pos[1] = pos_actual[nro_ciudades];
8.  FOR i := 2 TO nro_ciudades - 1 DO
9.     r := rand(0, 1);  /* número aleatorio entre 0 y 1 */
10.    ciudad_elegida = FALSE;
11.    IF(r <= K3) THEN
12.       Elegir la ciudad j que aparece luego de la ciudad indicada por
          nueva_pos[i-1] en mejor_global;
13.       IF(ciudad j no está en nueva_pos) THEN
14.          nueva_pos[i] := ciudad j;
15.          ciudad_elegida = TRUE;
16.       END_IF
17.    END_IF
18.    IF(r <= (K2 + K3) OR ciudad_elegida = FALSE) THEN
19.       Elegir la ciudad j que aparece luego de la ciudad indicada por
          nueva_pos[i-1] en mejor_local;
20.       IF(ciudad j no está en nueva_pos) THEN
21.          nueva_pos[i] := ciudad j;
22.          ciudad_elegida = TRUE;
23.       END_IF
24.    END_IF
25.    IF(ciudad_elegida = FALSE) THEN
26.       nueva_pos[i] := ultima ciudad en la pos_actual que no aparece
                          en nueva_pos;
27.    END_IF
28. END_FOR
29. nueva_pos[nro_ciudades] := ciudad restante;
30. retornar(nueva_pos);
```

Figura 4.18. Algoritmo PSO_AS para deducir una nueva posición para una partícula, a partir su posición actual, una mejor posición de su vecindad y una mejor posición del enjambre.

El algoritmo PSO_AS evita que una ciudad que ya se encuentra en la nueva posición para una partícula vuelva a ser elegida. De forma que, si la ciudad elegida de la mejor posición del enjambre ya se encuentra dentro de la nueva posición para la partícula, el algoritmo intenta elegir la siguiente ciudad de la mejor posición en la vecindad de la partícula. Si nuevamente resulta que la ciudad elegida de la mejor posición de la vecindad de la partícula ya se encuentra en la nueva posición, el algoritmo elige una ciudad de la posición actual de la partícula, caso en el cual no puede ocurrir que se elija una ciudad que ya se encuentre en la nueva posición que se está determinando.

En el algoritmo PSO_AS puede notarse que si un vértice *j* aparece luego de un vértice *i* en la mejor posición del enjambre y en la mejor posición de la vecindad, este vértice *j* tiene una probabilidad de $K_2 + K_3$ de aparecer luego del vértice *i* en la nueva posición para la partícula [Secrest01]. De allí el parecido de esta adaptación con los métodos ACO, en los cuales los

caminos entre ciudades más utilizados tienen mayor probabilidad de ser seleccionados nuevamente. Sin embargo, esto no significa que el algoritmo PSO_AS no sea válido para el PSO, puesto que este algoritmo conserva el concepto de utilizar la información proveniente de las mejores posiciones y de la posición actual de la partícula para determinar su nueva posición.

Además de lo anterior, también puede notarse que el algoritmo PSO_AS no utiliza ningún otro tipo de información adicional acerca del orden en que se deben visitar las ciudades para determinar la nueva posición de una partícula, sino que confía en que la nueva posición será buena, ya que ella se deduce a partir de buenas posiciones [Secrest01].

En las figuras a continuación se ilustra un ejemplo que muestra como se desarrolla el algoritmo PSO_AS.

Dado un problema TSP de 7 ciudades, con los parámetros $K_1 = 0.1$, $K_2 = 0.1$, $K_3 = 0.8$, la posición actual de la i-ésima partícula $X_i = [2, 4, 6, 5, 3, 7, 1]$, la mejor posición de su vecindad $L_i = [7, 6, 2, 3, 5, 4, 1]$, la mejor posición del enjambre G = [3, 7, 6, 4, 5, 1, 2] y los números generados al azar 0.6, 0.82, 0.87, 0.26, 0.94, la construcción de una nueva posición para la i-ésima partícula se obtiene como sigue:

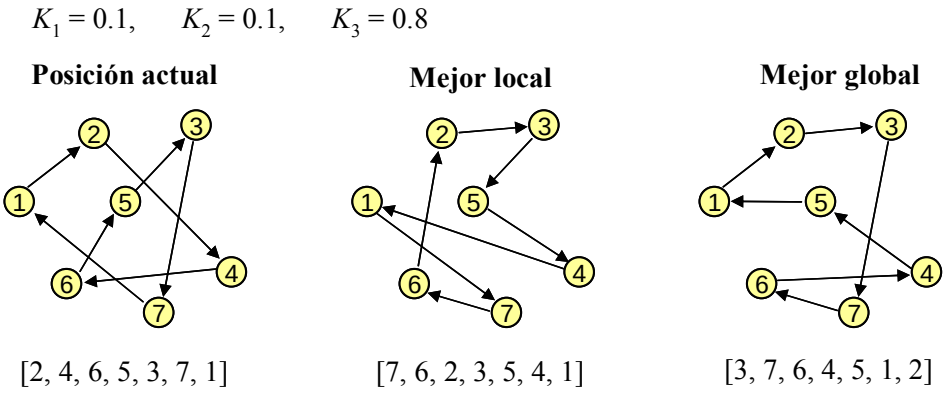


Figura 4.19. Parámetros iniciales para el algoritmo PSO_AS.

- Se elige como ciudad inicial de la nueva posición para la i-ésima partícula a la última ciudad de su posición actual. La ciudad 1 es elegida y establecida como la última ciudad determinada en la nueva posición, lo que se indica mediante la flecha sobre tal ciudad en la figura 4.20.

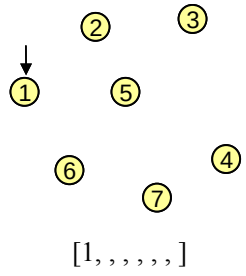
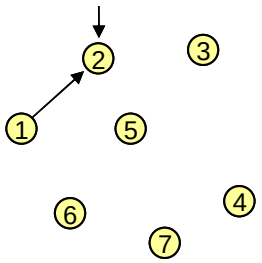


Figura 4.20. Elección de la ciudad inicial para la nueva posición de la i-ésima partícula.

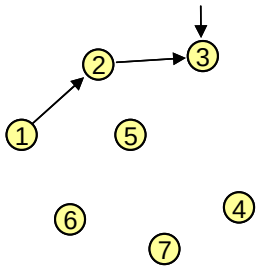
- Dado que el primer número aleatorio 0.6 es menor que K_3, la información de la mejor posición global es utilizada. La ciudad 2 es establecida como siguiente ciudad en la nueva posición para la *i*-ésima partícula ya que ella sigue a la ciudad 1 en la mejor posición global.



[1, 2, , , , ,]

Figura 4.21. Elección de la segunda ciudad para la nueva posición de la i-ésima partícula.

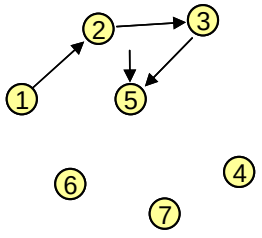
- Como el segundo número aleatorio 0.82 es mayor a K_3 y menor a $K_3 + K_2$, la información de la mejor posición en la vecindad de la partícula es utilizada. La ciudad 3 es elegida como siguiente ciudad en la nueva posición para la *i*-ésima partícula, ya que ella sigue a la ciudad 2 en la mejor posición de la vecindad.



[1, 2, 3, , , ,]

Figura 4.22. Elección de la tercera ciudad para la nueva posición de la i-ésima partícula.

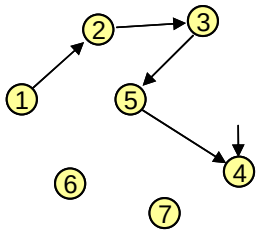
- Dado que el tercer número aleatorio 0.87 es mayor a K_3 y menor a $K_3 + K_2$, la información de la mejor posición del la vecindad es utilizada, eligiéndose a la ciudad 5 como la siguiente ciudad en la nueva posición para la *i*-ésima partícula.



[1, 2, 3, 5, , ,]

Figura 4.23. Elección de la cuarta ciudad para la nueva posición de la i-ésima partícula.

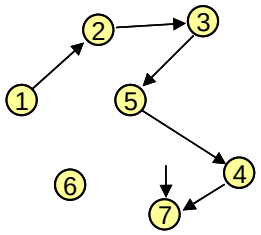
- Como el cuarto número aleatorio 0.26 es menor a K_3, la información de la mejor posición global es utilizada y esta indica que debe elegirse como siguiente ciudad a la ciudad 1, pero como esta ciudad ya no puede ser escogida debido a que ya ha sido incluida en la nueva posición, se opta por utilizar la información de la mejor posición de la vecindad y se elige como siguiente ciudad a la ciudad 4.



[1, 2, 3, 5, 4, ,]

Figura 4.24. Elección de la quinta ciudad para la nueva posición de la i-ésima partícula.

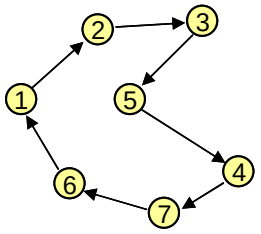
- Dado que el quinto número aleatorio 0.94 es mayor a $K_3 + K_2$, la información de la posición actual de la partícula es utilizada. La ciudad 7 es escogida ya que es la última ciudad en la posición actual de la *i*-ésima partícula que no aparece en la nueva posición.



[1, 2, 3, 5, 4, 7,]

Figura 4.25. Elección de la sexta ciudad para la nueva posición de la i-ésima partícula.

- Al ser la ciudad 6 la única ciudad no incluida en la nueva posición para la *i*-ésima partícula, esta es incluida. Finalmente la posición retornada como nueva posición para la partícula es [1, 2, 3, 5, 4, 7, 6].



[1, 2, 3, 5, 4, 7, 6]

Figura 4.26. Nueva posición para la partícula i retornada por el algoritmo PSO_AS.

Para utilizar el algoritmo PSO_AS dentro del algoritmo PSO o de los algoritmos MOPSOs, se reemplazan las sentencias correspondientes a las ecuaciones para el cálculo de la nueva velocidad (4.1) y de la nueva posición (4.2) de las partículas por la llamada al algoritmo PSO_AS.

En la figura 4.31 se ilustra el pseudocódigo basado en el algoritmo MOPSO genérico de la figura 4.6 para resolver un problema TSP multiobjetivo, que utiliza el algoritmo PSO_AS para determinar las nuevas posiciones de las partículas.

```
1.  Leer los parámetros K1, K2, K3, nro_objetivos, nro_ciudades y nro_partículas;
2.  Leer los costes de las arista para cada objetivo del problema TSP;
3.  FOR i := 1 TO nro_partículas DO
4.     Inicializar la partícula i a una posición aleatoria; /* permutación de ciudades */
5.     Evaluar la partícula i según la función objetivo del problema TSP;
6.     Actualizar el repositorio global;
7.     Actualizar los repositorios de las vecindades a las que pertenece la partícula;
8.  END_FOR
9.  WHILE(NOT condición_de_parada()) DO
10.    FOR i := 1 TO nro_de_partículas DO
11.       /* seleccionar mejor posición personal */
12.       mejor_local := seleccionar cualquier posición del repositorio de la vecindad
                         de la partícula;
13.       /* seleccionar mejor posición global */
14.       mejor_global := seleccionar una mejor posición global según el método MOPSO;
15.       /* Actualizar la posición de la partícula - llamada al algoritmo PSO_AS */
16.       partícula[i].pos := PSO_AS(partícula[i].pos, mejor_local, mejor_global,
                              K1, K2, K3, nro_ciudades)
17.    END_FOR
18.    FOR i := 1 TO nro_de_partículas DO
19.       Evaluar la partícula i según la función objetivo vectorial del problema TSP;
20.       Actualizar el repositorio global;
21.       Actualizar los repositorios de las vecindades a las que pertenece la partícula;
22.    END_FOR
23. END_WHILE
24. Retornar_soluciones_no_dominadas(rep_global);
```

Figura 4.27. Pseudocódigo de un algoritmo MOPSO genérico para resolver un problema TSP multiobjetivo.

Un requerimiento para la utilización del algoritmo PSO_AS consiste en la definición de un esquema de vecindades sobre el enjambre, de manera a obtener de la vecindad de cada partícula una mejor posición local a ser utilizada como parámetro en la llamada al algoritmo PSO_AS cada vez que se necesita determinar una nueva posición para las partículas.

El esquema de vecindades utilizado en las implementaciones de los métodos MOPSOs de este trabajo es el 2-*nearest* (véase sección 4.4.4). De esta forma, para cada partícula del enjambre se tiene un repositorio que almacena todas las soluciones no dominadas entre si encontradas por las partículas de su vecindad. Así, cada vez que se requiera de una mejor posición de la vecindad de la partícula para la determinación de su nueva posición, se toma a cualquier solución dentro del repositorio de su vecindad.

Además, el algoritmo PSO_AS también requiere de una mejor posición global para determinar la nueva posición de una partícula, en este caso se procede a seleccionar a una solu-

ción no dominada almacenada en el repositorio global de soluciones o en el repositorio personal de la partícula según lo indicado por cada método MOPSO.

La representación computacional de una partícula para la resolución del TSP multiobjetivo en las implementaciones de este trabajo incluye un vector de N componentes enteras para almacenar la posición de la partícula, la cual indica un recorrido hamiltoniano a través del grafo del problema TSP a resolver, y un vector M componentes para almacenar la evaluación correspondiente a la posición de la partícula en cada objetivo.

Los repositorios de soluciones no dominadas implementados en los distintos métodos MOPSO consisten básicamente en una lista de registros capaces de almacenar la estructura de datos de una partícula y de brindar eficientemente la funcionalidad requerida por cada método MOPSO en particular.

Capítulo 5

Resultados Experimentales

5.1 Ambiente de Ejecución

Para los efectos de este trabajo se realizaron implementaciones en C++ de los métodos MOPSOs propuestos por Moore y Chapman (mcMOPSO), Hu y Eberhart (heMOPSO), Coello y Lechuga (clMOPSO), Fieldsend y Singh (fsMOPSO), y Mostaghim y Teich (mtMOPSO) presentados en el capítulo 4, como de los métodos MOACOs propuestos por Barán y Schaerer (MOACS), Pinto y Barán (M3AS), y Doerner et al. (PACO) presentados en el capítulo 3.

Las implementaciones fueron compiladas utilizando el compilador GCC v3.2.3 (MinGW) y corridas en una máquina con procesador AMD Athlon64 3200+ de 2000MHz, con 512 MB de memoria RAM y sistema operativo Windows XP.

Luego de varias pruebas empíricas se encontraron que los valores 0, 0.05 y 0.95 resultaron apropiados para los parámetros K_1, K_2 y K_3 del algoritmo PSO_AS respectivamente, siendo el número de partículas utilizadas en los métodos MOPSOs igual a 10 y sin límite de tamaño para los repositorios de soluciones no dominadas.

Específicamente, para el método heMOPSO se estableció el valor del parámetro m en 2 y en el momento de actualizar la posición de cada partícula se eligió de manera aleatoria el objetivo de proximidad. Para el método clMOPSO el número de divisiones en cada objetivo se estableció en 20, lo cual da un total de 400 hipercubos.

Para los métodos MOACOs también se realizaron varias pruebas empíricas de manera a determinar los mejores valores para sus parámetros. El número de hormigas se estableció por analogía en 10, con los parámetros α, β, ρ, y τ_0 puestos a 1, 2, 0.1 y 1 respectivamente. En los métodos MOACS y PACO el valor para el parámetro q_0 fue establecido en 0.5. Para los métodos MOACS y M3AS los parámetros λ_1^i y λ_2^i de cada hormiga $i \in [1\ldots10]$ fueron establecidos tal que $\lambda_1^i = i$ y $\lambda_2^i = 10 - i + 1$.

Los problemas resueltos fueron determinados como combinaciones de 2 instancias TSP mono objetivas, obtenidas de la librería de instancias de prueba TSPLIB (http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95). Estas instancias son:

- KROA100, KROB100 y KROC100 de 100 ciudades cada una,
- KROA150 y KROB150 de 150 ciudades cada una, y
- KROA200 y KROB200 de 200 ciudades cada una.

Así, todos los métodos fueron utilizados para resolver 4 instancias TSP bi-objetivas, las cuales son:

- KROAB100 y KROAC100 de 100 ciudades cada una,
- KROAB150 de 150 ciudades, y
- KROAB200 de 200 ciudades.

5.2 Medida del Desempeño

Para comparar el desempeño de cada uno de los métodos implementados fueron utilizadas como referencia todas las soluciones no dominadas encontradas por el método PLS-3opt (véase sección 3.6.2) en las instancias KROAB100 y KROAC100, y por el método PD-TPLS 3.first.ils (véase sección 3.6.3) en los problemas KROAB150 y KROAB200, las cuales se encuentran disponibles en Internet (http://w3.ualg.pt/~lpaquete/tsp/). Para cada instancia fueron combinados los resultados de las diferentes corridas del método correspondiente y se extrajeron todas las soluciones no dominadas entre si, las cuales fueron consideradas como el frente Pareto real Y_{true} de cada problema.

Para poder evaluar los resultados obtenidos en cada corrida de los métodos MOPSOs y MOACOs fueron utilizadas las métricas M_1^*, M_2^* y M_3^* propuestas por Zitzler et al. [Ziztzler00], que evalúan respectivamente la *calidad de las soluciones*, la *distribución de las soluciones* y la *extensión* del frente Pareto aproximado *Y'* devuelto en cada corrida. También los métodos fueron comparados con respecto al número de soluciones no dominadas encontradas en cada corrida, denotado por $|Y'|$.

La métrica M_1^* proporciona una idea de la aproximación al frente Pareto real Y_{true} de un frente Pareto aproximado *Y'*, calculando el promedio de las distancias euclidianas de cada solución en el frente *Y'* a la solución más cercana en el frente Y_{true}.

La métrica M_2^* estima la distribución promedio de las soluciones a lo largo de un frente Pareto aproximado *Y'*, calculando el número promedio de soluciones que se encuentran separadas de cada solución a una distancia mayor que cierto valor σ definido a priori.

La métrica M_3^* evalúa la extensión o abarcamiento de un frente Pareto aproximado *Y'* a través de la sumatoria de las máximas separaciones de las evaluaciones en cada objetivo.

La métrica de cantidad de soluciones $|Y'|$ da una idea acerca de la diversidad de combinación de las evaluaciones de los objetivos presentadas al Tomador de Decisiones, esta métrica puede ser considerada como un complemento de las demás métricas.

En la figura 5.1 se presentan ejemplos de los criterios de comparación de frentes Pareto aproximados promovidos por las métricas M_1^*, M_2^* y M_3^* que a continuación se definen formalmente:

$$M_1^*(Y') = \frac{1}{|Y'|} \sum_{p \in Y'} \min \left(d(p,q); \forall q \in Y_{true} \right)$$

$$M_2^*(Y') = \frac{1}{|Y'|-1} \sum_{p \in Y'} \left| \{ q \mid q \in Y' \wedge d(p, q) > \sigma \} \right|$$

$$M_3^*(Y') = \sqrt{\sum_{i=1}^{M} max(|p_i - q_i|)} \quad \forall p, q \in Y'$$

donde Y' es el frente Pareto aproximado devuelto en una corrida, $d(p, q)$ calcula la distancia euclidiana entre las soluciones p y q, $|\cdot|$ representa la cardinalidad, M es la dimensión del espacio objetivo y σ se estableció al 10% de la distancia entre los puntos extremos del frente Pareto aproximado Y'.

Para cada corrida, los valores de sus evaluaciones en cada métrica fueron normalizados a un número en el intervalo [0, 1], de manera a poder utilizar estos resultados en rankings de métodos presentados en la siguiente sección.

Las evaluaciones en la métrica M_1^* fueron normalizadas restando de 1 el resultado de la división de la evaluación de cada corrida por el mayor valor obtenido en esta métrica en cada problema.

Para la métrica M_2^*, la evaluación máxima de una corrida es igual al número de soluciones no dominadas encontradas en dicha corrida [Zitzler00], así las evaluaciones de las corridas fueron normalizadas dividiéndolas por el número de soluciones encontradas en dichas corridas.

Con relación a la métrica M_3^*, las evaluaciones de cada corrida fueron normalizadas dividiéndolas por el valor de evaluación de esta métrica para el frente Pareto real Y_{true} de cada problema.

La cantidad de soluciones no dominadas encontradas $|Y'|$ en cada corrida fue normalizada dividiéndola por el mayor valor de evaluación de esta métrica en cada problema.

De esta forma, los valores de evaluación normalizados son siempre menores que 1 y se consideran mejores cuanto más próximos encuentren a dicho valor.

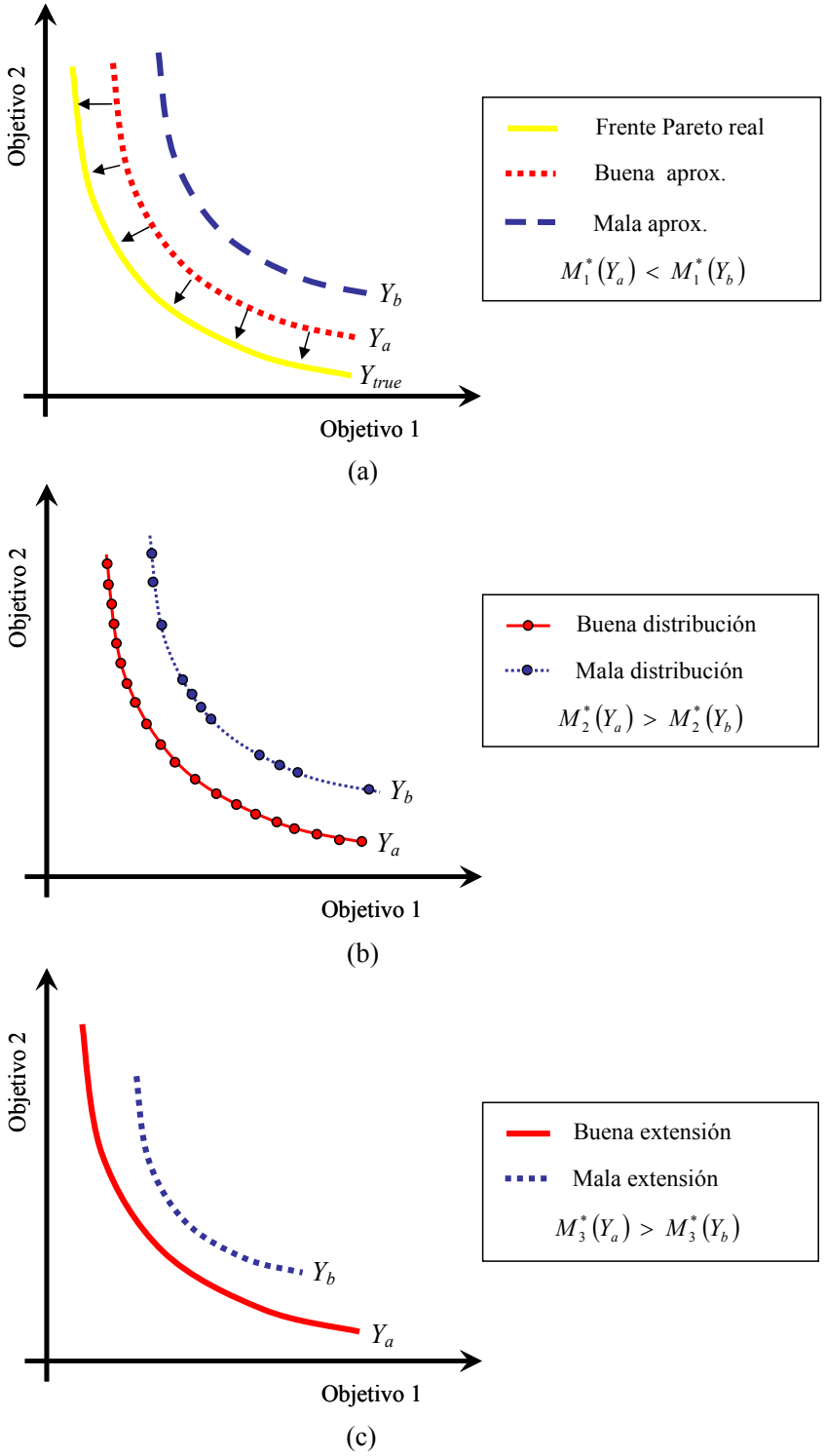


Figura 5.1. Ejemplos de los criterios de (a) calidad, (b) distribución de soluciones y (c) extensión para la comparación de frentes Pareto aproximados.

A continuación se presentan las ecuaciones para la normalización de los valores de evaluación de las métricas:

$$\left\| M_1^*(Y') \right\| = 1 - \frac{M_1^*(Y')}{M_{1MAX}^{*P}}$$

$$\left\| M_2^*(Y') \right\| = \frac{M_2^*(Y')}{|Y'|}$$

$$\left\| M_3^*(Y') \right\| = \frac{M_3^*(Y')}{M_3^*(Y_{true}^P)}$$

$$\left\| |Y'| \right\| = \frac{|Y'|}{|Y'|_{MAX}^P}.$$

donde *Y'* es el frente Pareto aproximado devuelto en una corrida, $|\cdot|$ representa la cardinalidad, $\|\cdot\|$ representa la normalización de la evaluación de una métrica, $M_{1\,MAX}^{*P}$ es el mayor valor de evaluación de la métrica M_1^* en el problema *P*, Y_{true}^P es el frente Pareto real del problema *P* y $|Y'|_{MAX}^P$ es el mayor valor de evaluación de la métrica de cantidad de soluciones en el problema *P*.

5.3 Resultados de las Corridas

Todas las implementaciones de los métodos MOPSOs y MOACOs fueron corridas 10 veces durante 200 segundos para cada problema de prueba, de manera a realizar un experimento justo y exhaustivo con cada uno de estos métodos, en cada problema resuelto.

Por cada problema y cada método utilizado se presenta una tabla que contiene las evaluaciones normalizadas de cada corrida. Al final de tabla se encuentran los valores promedios y las desviaciones estándar en cada métrica, de esta forma es posible conocer cual es el comportamiento promedio de los métodos en cada métrica y que tanto se diferencian sus evaluaciones entre una corrida y otra.

En los apartados siguientes se presentan los resultados obtenidos en cada problema resuelto y en el apartado final de esta sección se presentan las conclusiones generales derivadas a partir de la consideración de todos los resultados obtenidos.

5.3.1 Problema KROAB100

En las tablas 5.1 a 5.8 se presentan las evaluaciones normalizadas en las métricas M_1^*, M_2^*, M_3^* y $|Y'|$ para las 10 corridas de cada método en la resolución del problema KROAB100.

Para el mismo problema, en la tabla 5.9 se presenta un *ranking de métodos en cada métrica* a partir de sus evaluaciones promedio.

Puede verse que en este problema todos los métodos MOPSOs presentan mejores evaluaciones promedio en lo que se refiere a la calidad de las soluciones, superando a los métodos MOACOs y siendo el método mtMOPSO el mejor entre todos.

También, la mayoría de los métodos muestran un comportamiento promedio similar en lo que se refriere a la distribución de las soluciones, siendo los métodos MOACOs los que presentan mejores valores promedios de evaluación en esta métrica.

Por su parte, los métodos M3AS y MOACS presentan las mejores evaluaciones en lo que se refiere a la extensión del frente Pareto generado, seguidos por el método clMOPSO.

Por último, puede notarse que los métodos MOPSOs presentan mayor cantidad de soluciones no dominadas que los métodos MOACOs, siendo el método clMOPSO el mejor en esta métrica.

Las figuras 5.2 a 5.7 complementan gráficamente los resultados mostrados en la tabla 5.9. Para cada método, los resultados de todas sus corridas fueron combinados y se extrajeron las soluciones no dominadas entre si, de esta manera se obtiene el *mejor frente Pareto aproximado* de cada método. Así, por cada método MOPSO se elaboró una gráfica que muestra el frente Pareto real del problema KROAB100 y los mejores frentes Pareto aproximados de los tres métodos MOACOs y del método MOPSO correspondiente.

En las figuras 5.2 a 5.7 es posible observar que los métodos MOPSOs presentan mejores soluciones que los métodos MOACOs y a excepción del método mcMOPSO, todas las soluciones de los MOPSOs dominan a soluciones de los MOACOs, especialmente en la zona en donde ambos objetivos se optimizan con igual importancia (es decir, donde ambos objetivos tienen evaluación media). También se observa que los frentes Pareto aproximados de los métodos MOACS y M3AS pierden calidad a medida que se acercan a esta zona. De forma que, si se buscan soluciones que optimicen ambos criterios con la misma importancia, los métodos MOPSOs presentarán mejores resultados que los métodos MOACOs en este problema. Además, puede verse que los métodos MOACS, M3AS y clMOPSO presentan los frentes Pareto aproximados de mejor extensión, siendo los métodos MOACS y M3AS los únicos en conseguir soluciones próximas a los extremos del frente Pareto real.

También se nota, que los frentes Pareto aproximados correspondientes a los métodos MOPSOs son más densos en soluciones, mientras que los frentes Pareto aproximados de los métodos MOACOs presentan pequeños huecos entre sus soluciones.

clMOPSO - KROAB100				
Corrida	M_1^*	M_2^*	M_3^*	$\|\boldsymbol{Y}'\|$
1	0,80	0,82	0,87	1,00
2	0,79	0,81	0,86	0,98
3	0,81	0,81	0,88	0,96
4	0,78	0,79	0,84	0,95
5	0,72	0,81	0,89	0,91
6	0,80	0,81	0,88	0,91
7	0,77	0,82	0,88	0,90
8	0,77	0,81	0,89	0,89
9	0,84	0,81	0,87	0,88
10	0,76	0,81	0,91	0,87
Promedio	**0,78**	**0,81**	**0,88**	**0,93**
Desv. Est.	**0,03**	**0,01**	**0,02**	**0,04**

Tabla 5.1. Evaluaciones normalizadas de las corridas del método clMOPSO para el problema KROAB100.

fsMOPSO – KROAB100				
Corrida	M_1^*	M_2^*	M_3^*	$\|\boldsymbol{Y}'\|$
1	0,82	0,80	0,70	0,85
2	0,87	0,80	0,71	0,85
3	0,87	0,81	0,70	0,88
4	0,83	0,79	0,70	0,88
5	0,84	0,79	0,69	0,82
6	0,83	0,80	0,69	0,80
7	0,84	0,81	0,71	0,78
8	0,85	0,79	0,70	0,94
9	0,82	0,79	0,68	0,86
10	0,85	0,80	0,69	0,87
Promedio	**0,84**	**0,80**	**0,70**	**0,85**
Desv. Est.	**0,02**	**0,01**	**0,01**	**0,05**

Tabla 5.2. Evaluaciones normalizadas de las corridas del método fsMOPSO para el problema KROAB100.

heMOPSO - KROAB100				
Corrida	M_1^*	M_2^*	M_3^*	$\|\boldsymbol{Y}'\|$
1	0,93	0,74	0,53	0,95
2	0,89	0,74	0,55	0,84
3	0,91	0,74	0,52	0,84
4	0,88	0,72	0,51	0,89
5	0,86	0,75	0,53	0,79
6	0,87	0,73	0,55	0,82
7	0,95	0,73	0,54	0,80
8	0,93	0,74	0,51	0,77
9	0,85	0,72	0,53	0,88
10	0,89	0,69	0,53	0,88
Promedio	**0,90**	**0,73**	**0,53**	**0,85**
Desv. Est.	**0,03**	**0,02**	**0,01**	**0,06**

Tabla 5.3. Evaluaciones normalizadas de las corridas del método heMOPSO para el problema KROAB100.

mcMOPSO - KROAB100				
Corrida	M_1^*	M_2^*	M_3^*	$\|\boldsymbol{Y}'\|$
1	0,48	0,79	0,73	0,50
2	0,54	0,82	0,68	0,49
3	0,45	0,81	0,71	0,43
4	0,38	0,81	0,68	0,47
5	0,46	0,80	0,66	0,46
6	0,57	0,79	0,72	0,46
7	0,52	0,82	0,63	0,45
8	0,58	0,80	0,69	0,54
9	0,53	0,78	0,69	0,52
10	0,51	0,78	0,68	0,50
Promedio	**0,50**	**0,80**	**0,69**	**0,48**
Desv. Est.	**0,06**	**0,02**	**0,03**	**0,03**

Tabla 5.4. Evaluaciones normalizadas de las corridas del método mcMOPSO para el problema KROAB100.

mtMOPSO - KROAB100				
Corrida	M_1^*	M_2^*	M_3^*	$\|\boldsymbol{Y}'\|$
1	0,96	0,71	0,39	0,60
2	0,84	0,67	0,38	0,59
3	0,91	0,69	0,39	0,59
4	0,93	0,68	0,38	0,67
5	0,90	0,65	0,41	0,56
6	0,93	0,69	0,36	0,55
7	0,91	0,70	0,41	0,61
8	0,96	0,67	0,38	0,61
9	0,89	0,67	0,38	0,59
10	0,88	0,71	0,41	0,56
Promedio	**0,91**	**0,68**	**0,39**	**0,59**
Desv. Est.	**0,04**	**0,02**	**0,02**	**0,03**

Tabla 5.5. Evaluaciones normalizadas de las corridas del método mtMOPSO para el problema KROAB100.

MOACS - KROAB100				
Corrida	M_1^*	M_2^*	M_3^*	$\|\boldsymbol{Y}'\|$
1	0,00	0,82	0,87	0,18
2	0,16	0,83	0,92	0,22
3	0,18	0,82	0,92	0,24
4	0,20	0,82	0,90	0,24
5	0,21	0,82	0,90	0,24
6	0,21	0,81	0,90	0,22
7	0,20	0,82	0,90	0,21
8	0,24	0,81	0,90	0,24
9	0,22	0,82	0,90	0,25
10	0,25	0,82	0,90	0,25
Promedio	**0,19**	**0,82**	**0,90**	**0,23**
Desv. Est.	**0,07**	**0,01**	**0,01**	**0,02**

Tabla 5.6. Evaluaciones normalizadas de las corridas del método MOACS para el problema KROAB100.

M3AS - KROAB100				
Corrida	M_1^*	M_2^*	M_3^*	$\|\boldsymbol{Y}'\|$
1	0,19	0,81	0,94	0,20
2	0,31	0,83	0,94	0,19
3	0,17	0,81	0,94	0,23
4	0,17	0,80	0,94	0,22
5	0,19	0,82	0,92	0,20
6	0,19	0,82	0,92	0,24
7	0,27	0,81	0,94	0,23
8	0,31	0,84	0,93	0,19
9	0,30	0,82	0,95	0,19
10	0,27	0,83	0,93	0,22
Promedio	**0,24**	**0,82**	**0,94**	**0,21**
Desv. Est.	**0,06**	**0,01**	**0,01**	**0,02**

Tabla 5.7. Evaluaciones normalizadas de las corridas del método M3AS para el problema KROAB100.

PACO - KROAB100				
Corrida	M_1^*	M_2^*	M_3^*	$\|\boldsymbol{Y}'\|$
1	0,35	0,82	0,37	0,06
2	0,33	0,88	0,31	0,03
3	0,39	0,86	0,33	0,04
4	0,33	0,81	0,38	0,05
5	0,40	0,83	0,37	0,05
6	0,35	0,78	0,35	0,05
7	0,35	0,84	0,33	0,05
8	0,34	0,85	0,35	0,04
9	0,47	0,87	0,33	0,03
10	0,35	0,83	0,35	0,04
Promedio	**0,37**	**0,84**	**0,35**	**0,04**
Desv. Est.	**0,04**	**0,03**	**0,02**	**0,01**

Tabla 5.8. Evaluaciones normalizadas de las corridas del método PACO para el problema KROAB100.

Ranking KROAB100								
Pos.	M_1^*		M_2^*		M_3^*		$\|Y'\|$	
1º	mtMOPSO	0,91	PACO	0,84	M3AS	0,94	clMOPSO	0,93
2º	heMOPSO	0,90	M3AS	0,82	MOACS	0,90	fsMOPSO	0,85
3º	fsMOPSO	0,84	MOACS	0,82	clMOPSO	0,88	heMOPSO	0,85
4º	clMOPSO	0,78	clMOPSO	0,81	fsMOPSO	0,70	mtMOPSO	0,59
5º	mcMOPSO	0,50	fsMOPSO	0,80	mcMOPSO	0,69	mcMOPSO	0,48
6º	PACO	0,37	mcMOPSO	0,80	heMOPSO	0,53	MOACS	0,23
7º	M3AS	0,24	heMOPSO	0,73	mtMOPSO	0,39	M3AS	0,21
8º	MOACS	0,19	mtMOPSO	0,68	PACO	0,35	PACO	0,04

Tabla 5.9. Ranking de métodos en cada métrica para el problema KROAB100.

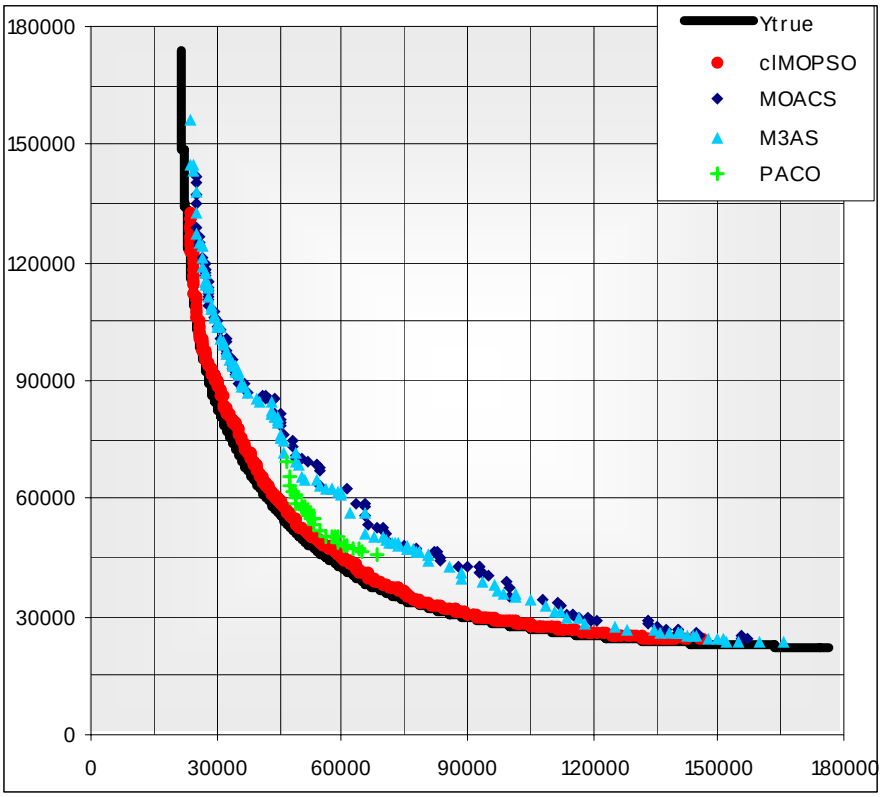


Figura 5.2. Mejores frentes Pareto aproximados del método clMOPSO y de los métodos MOACOs para el problema KROAB100.

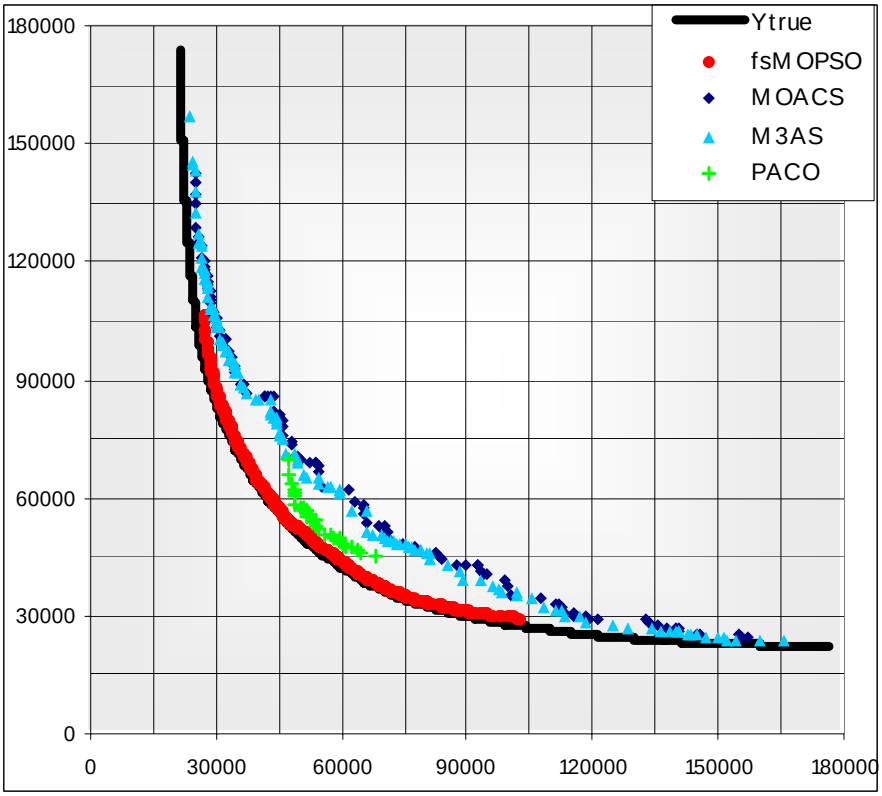


Figura 5.3. Mejores frentes Pareto aproximados del método fsMOPSO y de los métodos MOACOs para el problema KROAB100.

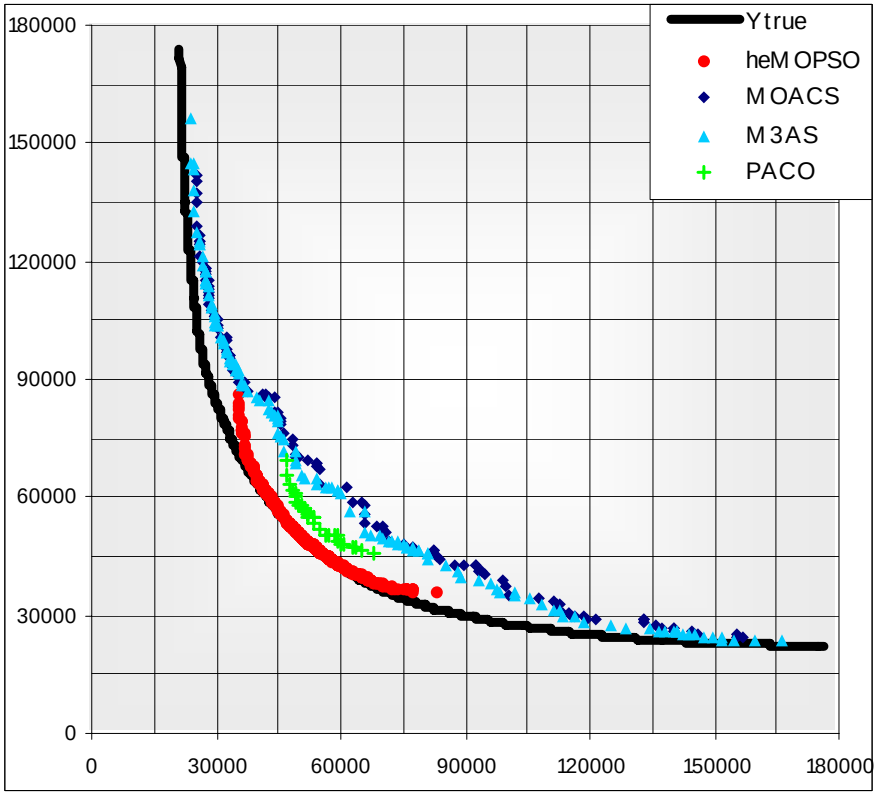


Figura 5.4. Mejores frentes Pareto aproximados del método heMOPSO y de los métodos MOACOs para el problema KROAB100.

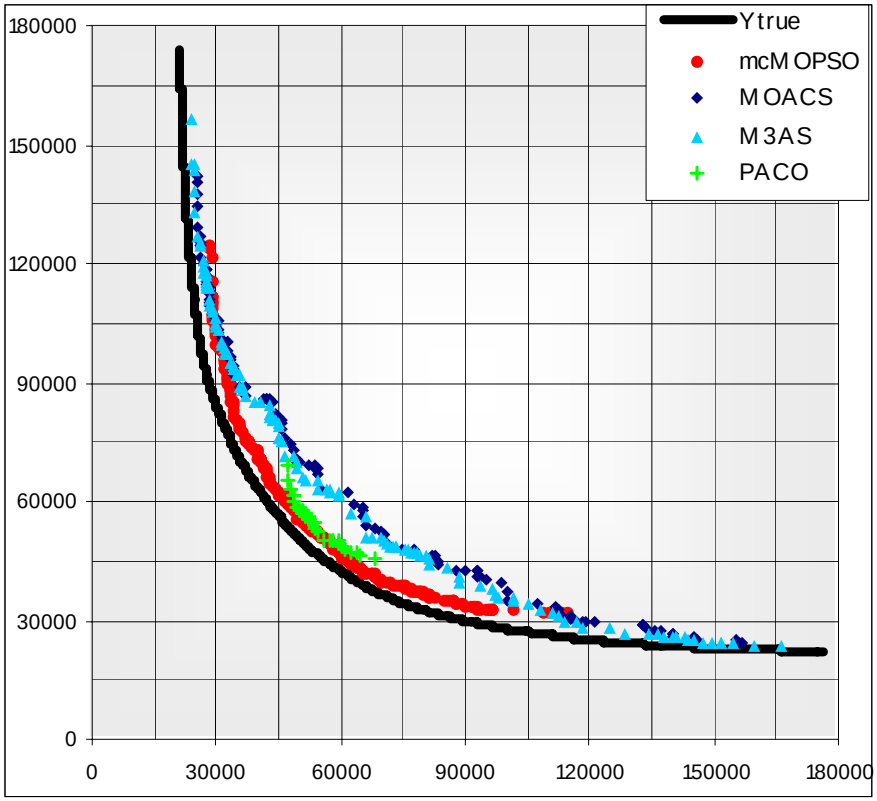


Figura 5.5. Mejores frentes Pareto aproximados del método mcMOPSO y de los métodos MOACOs para el problema KROAB100.

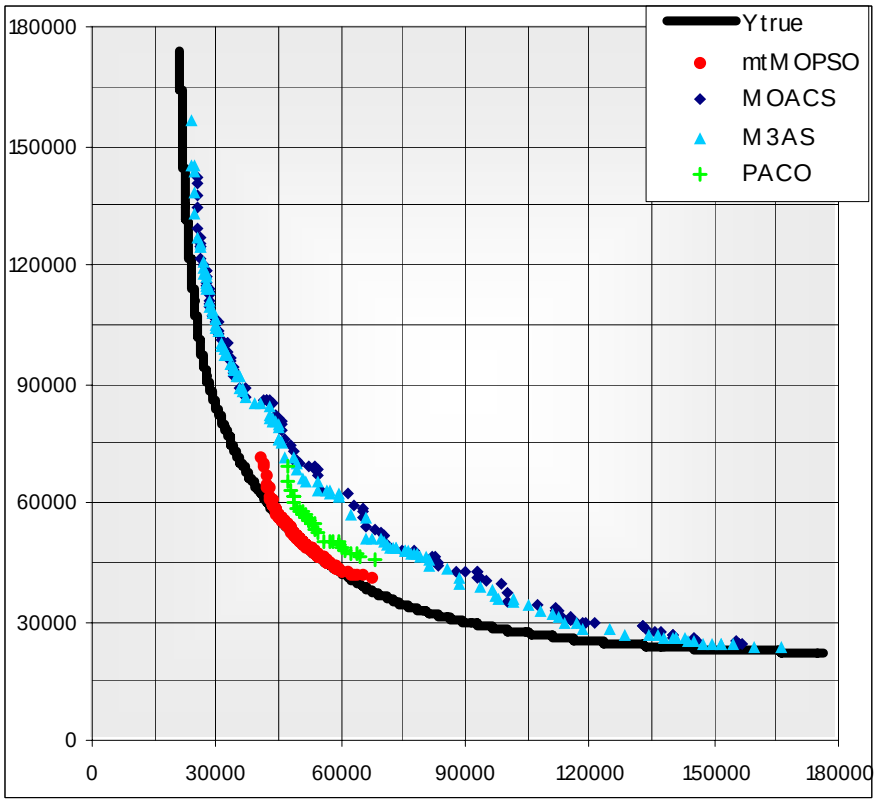


Figura 5.6. Mejores frentes Pareto aproximados del método mtMOPSO y de los métodos MOACOs para el problema KROAB100.

5.3.2 Problema KROAC100

Para el problema KROAC100 en las tablas 5.10 a 5.17 se presentan las evaluaciones promedio normalizadas de las 10 corridas por cada método. En la tabla 5.18 se presenta el ranking promedio de métodos en cada métrica y en las figuras 5.6 a 5.10 se complementa esta información con las gráficas de los mejores frentes Pareto aproximados confeccionadas por cada método MOPSO.

Se puede observar que para el problema KROAC100 se repiten las conclusiones parciales derivadas para el problema KROAB100.

Los métodos MOPSOs se aproximan más al frente Pareto real que los métodos MOACOs, siendo el método mtMOPSO el mejor y para la mayoría de los métodos MOPSOs todas sus soluciones dominan a soluciones de los métodos MOACOs.

En tanto que, en términos de distribución de soluciones casi todos los métodos presentan una evaluación promedio similar, siendo nuevamente los métodos MOACOs los mejores en esta métrica.

Además, los métodos MOACS y M3AS presentan los frentes Pareto aproximados de mayor extensión, seguidos del método clMOPSO y quedando últimos los métodos mtMOPSO y PACO. Siendo los métodos MOACS y M3AS los únicos en encontrar soluciones hacia los extremos del frente Pareto real de este problema.

Por último, Los métodos MOPSOs presentan mayor densidad y cantidad de soluciones en sus mejores frentes Pareto aproximados que los métodos MOACOs.

clMOPSO - KROAC100				
Corrida	M_1^*	M_2^*	M_3^*	$\|\boldsymbol{Y}'\|$
1	0,80	0,81	0,86	0,99
2	0,83	0,81	0,86	0,97
3	0,72	0,81	0,86	1,00
4	0,77	0,81	0,88	0,95
5	0,83	0,80	0,90	0,93
6	0,73	0,81	0,89	0,93
7	0,81	0,81	0,81	0,93
8	0,80	0,79	0,89	0,92
9	0,77	0,81	0,87	0,92
10	0,77	0,80	0,87	0,91
Promedio	**0,78**	**0,81**	**0,87**	**0,94**
Desv. Est.	**0,04**	**0,01**	**0,03**	**0,03**

Tabla 5.10. Evaluaciones normalizadas de las corridas del método clMOPSO para el problema KROAC100.

fsMOPSO - KROAC100				
Corrida	M_1^*	M_2^*	M_3^*	$\|\boldsymbol{Y}'\|$
1	0,84	0,78	0,70	0,87
2	0,78	0,77	0,70	0,83
3	0,75	0,78	0,69	0,91
4	0,81	0,78	0,69	0,82
5	0,84	0,78	0,70	0,85
6	0,81	0,77	0,69	0,84
7	0,84	0,80	0,68	0,95
8	0,81	0,79	0,72	0,82
9	0,85	0,80	0,70	0,86
10	0,82	0,80	0,70	0,85
Promedio	**0,81**	**0,79**	**0,70**	**0,86**
Desv. Est.	**0,03**	**0,01**	**0,01**	**0,04**

Tabla 5.11. Evaluaciones normalizadas de las corridas del método fsMOPSO para el problema KROAC100.

heMOPSO - KROAC100				
Corrida	M_1^*	M_2^*	M_3^*	$\|\boldsymbol{Y}'\|$
1	0,89	0,74	0,54	0,87
2	0,91	0,74	0,54	0,86
3	0,88	0,70	0,51	0,91
4	0,84	0,75	0,55	0,92
5	0,88	0,75	0,57	0,78
6	0,92	0,74	0,55	0,81
7	0,83	0,76	0,54	0,77
8	0,86	0,73	0,55	0,81
9	0,86	0,72	0,53	0,84
10	0,90	0,76	0,53	0,84
Promedio	**0,88**	**0,74**	**0,54**	**0,84**
Desv. Est.	**0,03**	**0,02**	**0,01**	**0,05**

Tabla 5.12. Evaluaciones normalizadas de las corridas del método heMOPSO para el problema KROAC100.

mcMOPSO - KROAC100				
Corrida	M_1^*	M_2^*	M_3^*	$\|\boldsymbol{Y}'\|$
1	0,53	0,78	0,69	0,50
2	0,49	0,80	0,71	0,50
3	0,45	0,79	0,65	0,47
4	0,51	0,79	0,68	0,46
5	0,50	0,76	0,70	0,53
6	0,43	0,78	0,69	0,52
7	0,52	0,79	0,71	0,45
8	0,46	0,81	0,70	0,49
9	0,53	0,77	0,69	0,49
10	0,42	0,80	0,72	0,51
Promedio	**0,48**	**0,79**	**0,69**	**0,49**
Desv. Est.	**0,04**	**0,02**	**0,02**	**0,03**

Tabla 5.13. Evaluaciones normalizadas de las corridas del método mcMOPSO para el problema KROAC100

mtMOPSO - KROAC100				
Corrida	M_1^*	M_2^*	M_3^*	$\|\boldsymbol{Y}'\|$
1	0,80	0,67	0,42	0,64
2	0,87	0,67	0,43	0,64
3	0,92	0,56	0,41	0,98
4	0,84	0,68	0,40	0,69
5	0,82	0,63	0,41	0,69
6	0,86	0,69	0,42	0,63
7	0,95	0,72	0,40	0,56
8	0,88	0,67	0,41	0,55
9	0,92	0,76	0,39	0,65
10	0,93	0,71	0,40	0,54
Promedio	**0,88**	**0,68**	**0,41**	**0,66**
Desv. Est.	**0,05**	**0,05**	**0,01**	**0,13**

Tabla 5.14. Evaluaciones normalizadas de las corridas del método mtMOPSO para el problema KROAC100.

MOACS - KROAC100				
Corrida	M_1^*	M_2^*	M_3^*	$\|\boldsymbol{Y}'\|$
1	0,00	0,82	0,91	0,20
2	0,07	0,83	0,92	0,20
3	0,14	0,83	0,91	0,22
4	0,14	0,83	0,91	0,24
5	0,10	0,83	0,91	0,24
6	0,10	0,83	0,91	0,24
7	0,15	0,82	0,91	0,26
8	0,14	0,82	0,91	0,26
9	0,13	0,82	0,91	0,27
10	0,14	0,82	0,91	0,27
Promedio	**0,11**	**0,82**	**0,91**	**0,24**
Desv. Est.	**0,05**	**0,01**	**0,00**	**0,03**

Tabla 5.15. Evaluaciones normalizadas de las corridas del método MOACS para el problema KROAC100.

M3AS - KROAC100				
Corrida	M_1^*	M_2^*	M_3^*	$\|\boldsymbol{Y}'\|$
1	0,12	0,82	0,90	0,23
2	0,19	0,83	0,92	0,23
3	0,20	0,82	0,92	0,21
4	0,21	0,83	0,93	0,24
5	0,11	0,82	0,92	0,22
6	0,16	0,83	0,92	0,19
7	0,20	0,84	0,91	0,17
8	0,16	0,82	0,92	0,22
9	0,14	0,81	0,93	0,22
10	0,17	0,82	0,92	0,21
Promedio	**0,16**	**0,82**	**0,92**	**0,21**
Desv. Est.	**0,04**	**0,01**	**0,01**	**0,02**

Tabla 5.16. Evaluaciones normalizadas de las corridas del método M3AS para el problema KROAC100.

PACO - KROAC100				
Corrida	M_1^*	M_2^*	M_3^*	$\|\boldsymbol{Y}'\|$
1	0,39	0,84	0,37	0,05
2	0,38	0,79	0,37	0,05
3	0,55	0,81	0,38	0,03
4	0,48	0,79	0,37	0,04
5	0,35	0,83	0,36	0,05
6	0,38	0,79	0,34	0,06
7	0,51	0,86	0,29	0,03
8	0,37	0,82	0,37	0,04
9	0,41	0,77	0,27	0,05
10	0,36	0,82	0,39	0,05
Promedio	**0,42**	**0,81**	**0,35**	**0,04**
Desv. Est.	**0,07**	**0,03**	**0,04**	**0,01**

Tabla 5.17. Evaluaciones normalizadas de las corridas del método PACO para el problema KROAC100.

Ranking KROAC100								
Pos.	M_1^*		M_2^*		M_3^*		$\|Y'\|$	
1º	mtMOPSO	0,88	MOACS	0,82	M3AS	0,92	clMOPSO	0,94
2º	heMOPSO	0,88	M3AS	0,82	MOACS	0,91	fsMOPSO	0,86
3º	fsMOPSO	0,81	PACO	0,81	clMOPSO	0,87	heMOPSO	0,84
4º	clMOPSO	0,78	clMOPSO	0,81	fsMOPSO	0,70	mtMOPSO	0,66
5º	mcMOPSO	0,48	mcMOPSO	0,79	mcMOPSO	0,69	mcMOPSO	0,49
6º	PACO	0,42	fsMOPSO	0,79	heMOPSO	0,54	MOACS	0,24
7º	M3AS	0,16	heMOPSO	0,74	mtMOPSO	0,41	M3AS	0,21
8º	MOACS	0,11	mtMOPSO	0,68	PACO	0,35	PACO	0,04

Tabla 5.18. Ranking de métodos por cada métrica en el problema KROAC100.

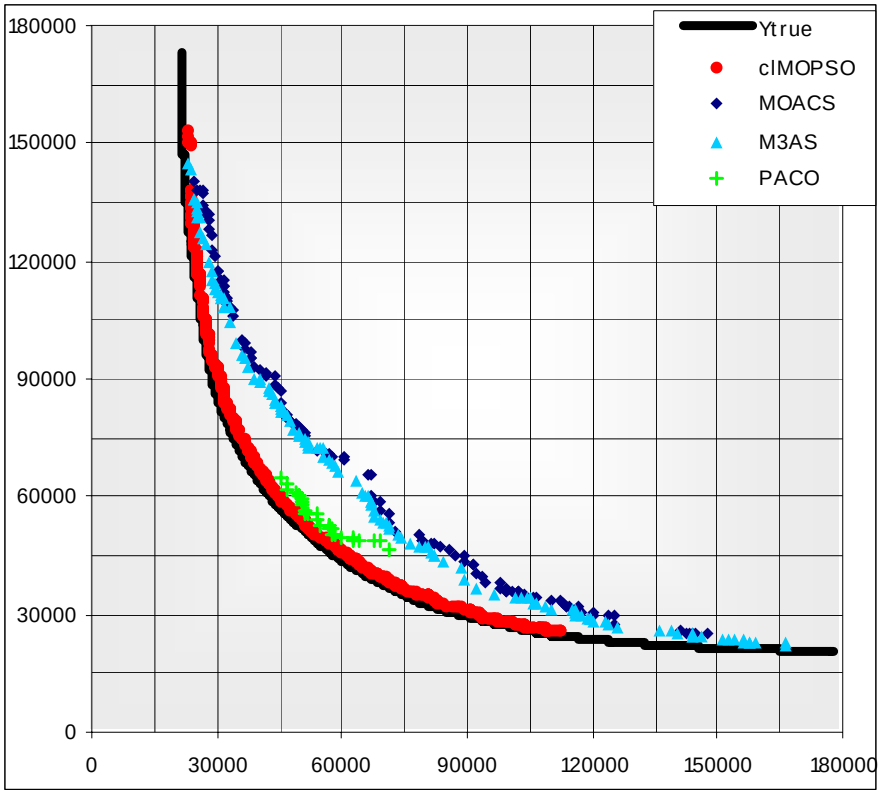


Figura 5.7. Mejores frentes Pareto aproximados del método clMOPSO y de los métodos MOACOs para el problema KROAC100.

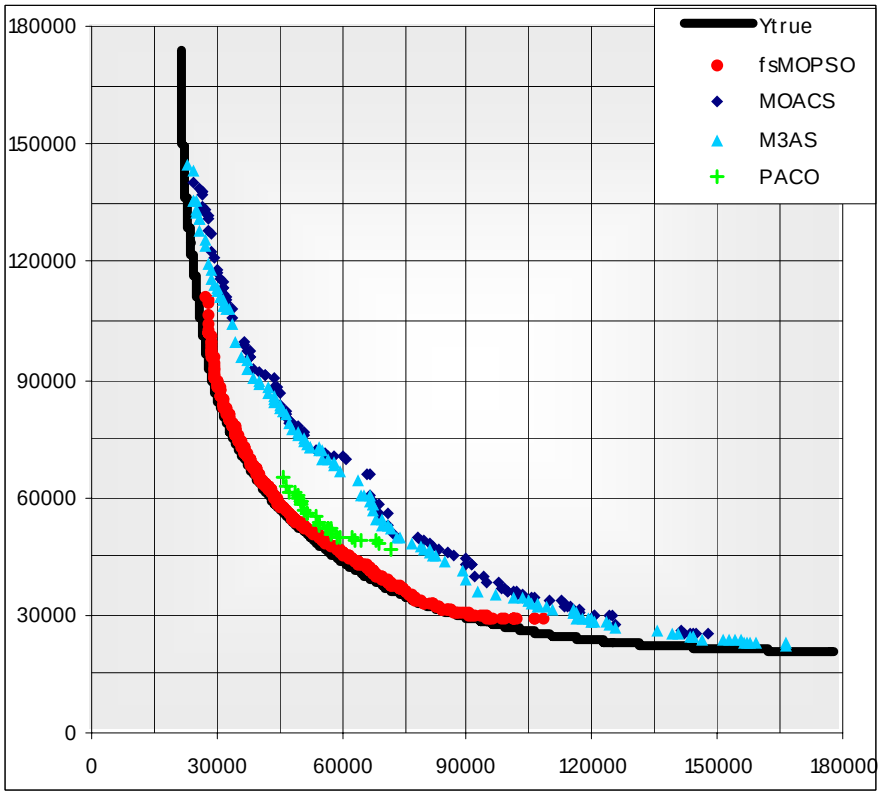


Figura 5.8. Mejores frentes Pareto aproximados del método fsMOPSO y de los métodos MOACOs para el problema KROAC100.

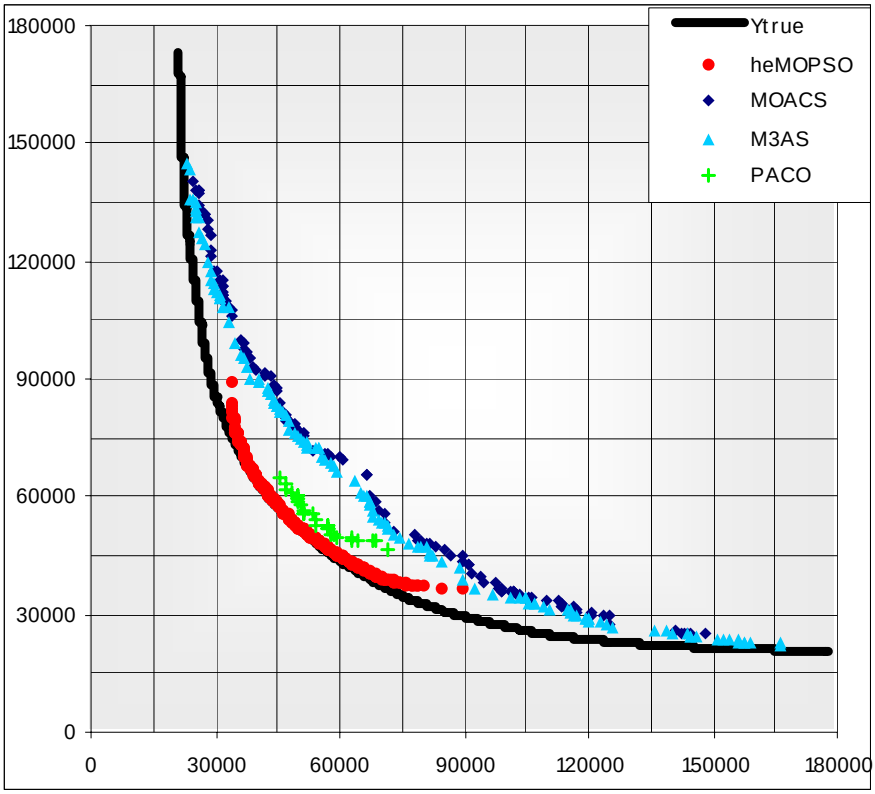


Figura 5.9. Mejores frentes Pareto aproximados del método heMOPSO y de los métodos MOACOs para el problema KROAC100.

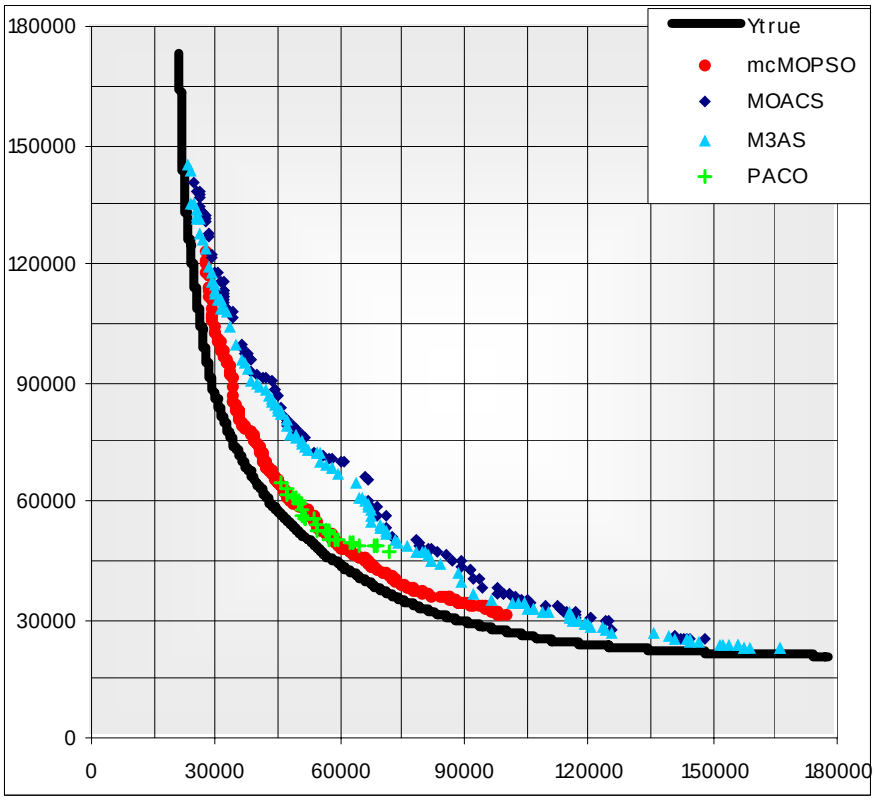


Figura 5.10. Mejores frentes Pareto aproximados del método mcMOPSO y de los métodos MOACOs para el problema KROAC100.

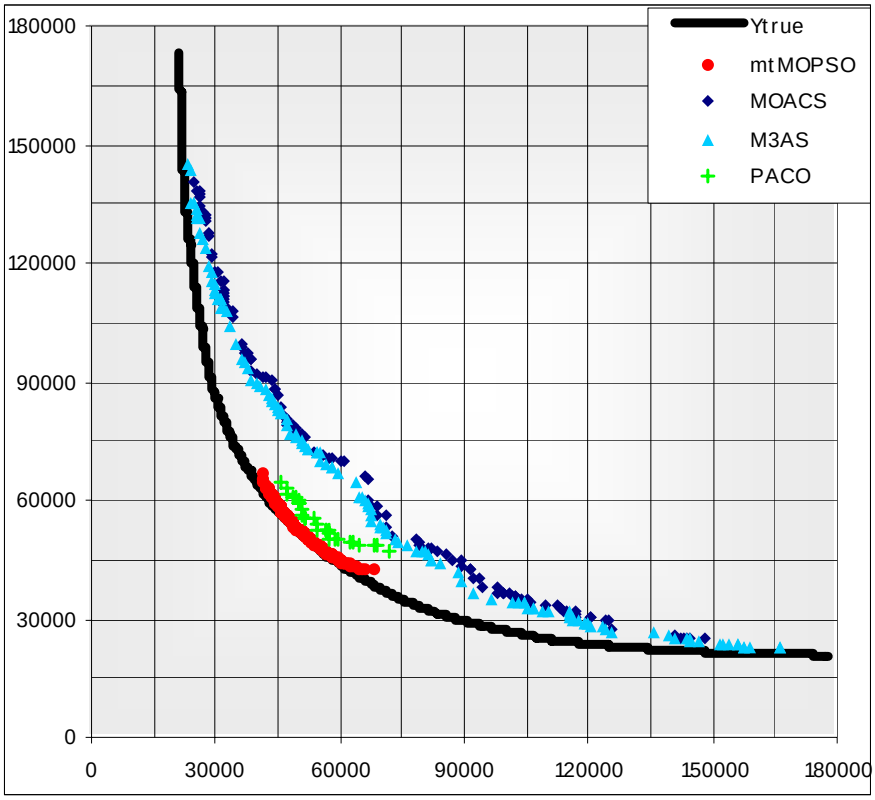


Figura 5.11. Mejores frentes Pareto aproximados del método mtMOPSO y de los métodos MOACOs para el problema KROAC100.

5.3.3 Problema KROAB150

Para el problema KROAB150 las evaluaciones normalizadas de las 10 corridas por cada método se muestran en las tablas 5.19 a 5.26, el ranking de evaluaciones promedio en cada métrica se presenta en la tabla 5.27 y en las figuras 5.12 a 5.16 se ilustran los mejores frentes Pareto aproximados por cada método MOPSO.

En este problema puede observarse una variación en las evaluaciones promedio en la métrica M_1^*. Los métodos mtMOPSO y heMOPSO presentan una evaluación similar a la de los casos anteriores, mientras que para los demás métodos sus evaluaciones disminuyen en gran proporción, lo cual se debe al aumento de la complejidad de resolución por el incremento del número de ciudades.

En la métrica M_2^*, las evaluaciones promedio varían levemente con respecto a los problemas anteriores, presentando nuevamente la mayoría de los métodos evaluaciones muy similares y siendo nuevamente los métodos MOACOs los mejores en esta métrica.

Para la métrica M_3^* también se observa una disminución considerable en la evaluación promedio de algunos métodos MOPSOs. Quedando en las tres primeras posiciones del ran-

king de esta métrica los métodos M3AS, MOACS y clMOPSO, así como en los problemas anteriores.

En cuanto a la cantidad de soluciones promedio en cada frente Pareto aproximado, el ranking permanece prácticamente igual que en los problemas anteriores. Presentando siempre los métodos MOPSOs un frente Pareto aproximado de mayor densidad y cantidad de soluciones en comparación a los frentes Pareto aproximados por los métodos MOACOs.

clMOPSO - KROAB150				
Corrida	M_1^*	M_2^*	M_3^*	$\|\boldsymbol{Y}'\|$
1	0,46	0,80	0,81	1,00
2	0,47	0,80	0,80	0,98
3	0,54	0,81	0,75	0,99
4	0,58	0,81	0,73	0,98
5	0,49	0,80	0,82	0,95
6	0,56	0,81	0,80	0,94
7	0,48	0,79	0,80	0,94
8	0,51	0,81	0,77	0,92
9	0,56	0,79	0,80	0,91
10	0,50	0,81	0,79	0,91
Promedio	**0,52**	**0,80**	**0,79**	**0,95**
Desv. Est.	**0,04**	**0,01**	**0,03**	**0,03**

Tabla 5.19. Evaluaciones normalizadas de las corridas del método clMOPSO para el problema KROAB150.

fsMOPSO - KROAB150				
Corrida	M_1^*	M_2^*	M_3^*	$\|\boldsymbol{Y}'\|$
1	0,59	0,80	0,64	0,92
2	0,68	0,81	0,63	0,88
3	0,69	0,81	0,63	0,86
4	0,70	0,81	0,60	0,84
5	0,67	0,81	0,63	0,80
6	0,58	0,81	0,66	0,88
7	0,70	0,80	0,63	0,86
8	0,66	0,81	0,63	0,82
9	0,69	0,80	0,65	0,81
10	0,66	0,81	0,65	0,80
Promedio	**0,66**	**0,81**	**0,64**	**0,85**
Desv. Est.	**0,04**	**0,00**	**0,02**	**0,04**

Tabla 5.20. Evaluaciones normalizadas de las corridas del método fsMOPSO para el problema KROAB150.

heMOPSO - KROAB150				
Corrida	M_1^*	M_2^*	M_3^*	$\|\boldsymbol{Y}'\|$
1	0,88	0,75	0,41	0,71
2	0,76	0,71	0,45	0,73
3	0,77	0,76	0,41	0,76
4	0,89	0,75	0,43	0,93
5	0,89	0,71	0,43	0,73
6	0,82	0,73	0,44	0,85
7	0,84	0,74	0,46	0,74
8	0,84	0,70	0,43	1,04
9	0,86	0,76	0,44	0,72
10	0,80	0,77	0,47	0,75
Promedio	**0,84**	**0,74**	**0,44**	**0,80**
Desv. Est.	**0,05**	**0,02**	**0,02**	**0,05**

Tabla 5.21. Evaluaciones normalizadas de las corridas del método heMOPSO para el problema KROAB150.

mcMOPSO - KROAB150				
Corrida	M_1^*	M_2^*	M_3^*	$\|\boldsymbol{Y}'\|$
1	0,06	0,81	0,69	0,64
2	0,13	0,80	0,67	0,61
3	0,08	0,81	0,70	0,60
4	0,06	0,80	0,65	0,60
5	0,20	0,81	0,66	0,59
6	0,02	0,80	0,68	0,58
7	0,20	0,79	0,66	0,58
8	0,15	0,80	0,66	0,58
9	0,01	0,81	0,70	0,57
10	0,01	0,81	0,70	0,57
Promedio	**0,09**	**0,80**	**0,68**	**0,59**
Desv. Est.	**0,07**	**0,01**	**0,02**	**0,02**

Tabla 5.22. Evaluaciones normalizadas de las corridas del método mcMOPSO para el problema KROAB150.

mtMOPSO - KROAB150				
Corrida	M_1^*	M_2^*	M_3^*	$\|\boldsymbol{Y}'\|$
1	0,90	0,65	0,35	0,76
2	0,89	0,70	0,31	0,73
3	0,90	0,70	0,33	0,65
4	0,88	0,67	0,33	0,64
5	0,90	0,66	0,33	0,86
6	0,89	0,60	0,36	0,82
7	0,86	0,67	0,34	0,77
8	0,86	0,66	0,34	0,63
9	0,95	0,68	0,33	0,63
10	0,92	0,63	0,37	0,62
Promedio	**0,89**	**0,66**	**0,34**	**0,71**
Desv. Est.	**0,03**	**0,03**	**0,02**	**0,09**

Tabla 5.23. Evaluaciones normalizadas de las corridas del método mtMOPSO para el problema KROAB150.

MOACS - KROAB150				
Corrida	M_1^*	M_2^*	M_3^*	$\|\boldsymbol{Y}'\|$
1	0,00	0,83	0,94	0,24
2	0,09	0,81	0,92	0,29
3	0,11	0,82	0,92	0,35
4	0,11	0,82	0,92	0,37
5	0,08	0,80	0,94	0,24
6	0,07	0,83	0,92	0,37
7	0,09	0,82	0,92	0,35
8	0,09	0,83	0,92	0,34
9	0,09	0,82	0,92	0,30
10	0,11	0,83	0,92	0,37
Promedio	**0,08**	**0,82**	**0,92**	**0,32**
Desv. Est.	**0,03**	**0,01**	**0,01**	**0,05**

Tabla 5.24. Evaluaciones normalizadas de las corridas del método MOACS para el problema KROAB150.

M3AS - KROAB150				
Corrida	M_1^*	M_2^*	M_3^*	$\|\boldsymbol{Y}'\|$
1	0,12	0,82	0,94	0,22
2	0,07	0,82	0,94	0,27
3	0,12	0,82	0,92	0,30
4	0,06	0,82	0,94	0,27
5	0,06	0,82	0,94	0,24
6	0,12	0,82	0,92	0,31
7	0,08	0,82	0,92	0,27
8	0,11	0,82	0,92	0,30
9	0,09	0,82	0,92	0,29
10	0,03	0,82	0,94	0,25
Promedio	**0,08**	**0,82**	**0,93**	**0,27**
Desv. Est.	**0,03**	**0,00**	**0,01**	**0,03**

Tabla 5.25. Evaluaciones normalizadas de las corridas del método M3AS para el problema KROAB150.

PACO - KROAB150				
Corrida	M_1^*	M_2^*	M_3^*	$\|\boldsymbol{Y}'\|$
1	0,23	0,87	0,29	0,04
2	0,26	0,81	0,28	0,05
3	0,26	0,82	0,29	0,05
4	0,27	0,79	0,29	0,05
5	0,26	0,81	0,32	0,06
6	0,26	0,81	0,32	0,06
7	0,27	0,81	0,32	0,06
8	0,27	0,82	0,32	0,06
9	0,27	0,85	0,32	0,06
10	0,28	0,82	0,32	0,06
Promedio	**0,26**	**0,82**	**0,31**	**0,05**
Desv. Est.	**0,01**	**0,02**	**0,02**	**0,01**

Tabla 5.26. Evaluaciones normalizadas de las corridas del método PACO para el problema KROAB150.

Ranking KROAB150								
Pos.	M_1^*		M_2^*		M_3^*		$\|Y'\|$	
1º	mtMOPSO	0,89	M3AS	0,82	M3AS	0,93	clMOPSO	0,95
2º	heMOPSO	0,84	PACO	0,82	MOACS	0,92	fsMOPSO	0,85
3º	fsMOPSO	0,66	MOACS	0,82	clMOPSO	0,79	heMOPSO	0,80
4º	clMOPSO	0,52	fsMOPSO	0,81	mcMOPSO	0,68	mtMOPSO	0,71
5º	PACO	0,26	mcMOPSO	0,80	fsMOPSO	0,64	mcMOPSO	0,59
6º	mcMOPSO	0,09	clMOPSO	0,80	heMOPSO	0,44	MOACS	0,32
7º	M3AS	0,16	heMOPSO	0,74	mtMOPSO	0,41	M3AS	0,21
8º	MOACS	0,11	mtMOPSO	0,68	PACO	0,35	PACO	0,04

Tabla 5.27. Ranking de métodos por cada métrica en el problema KROAB150.

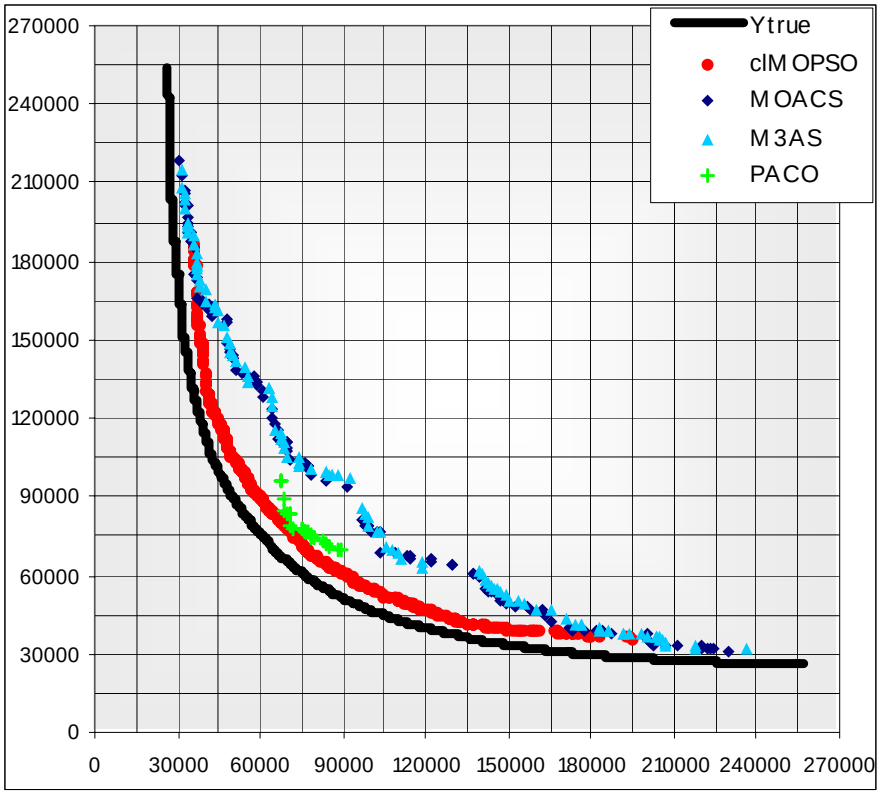


Figura 5.12. Mejores frentes Pareto aproximados del método clMOPSO y de los métodos MOACOs para el problema KROAB150.

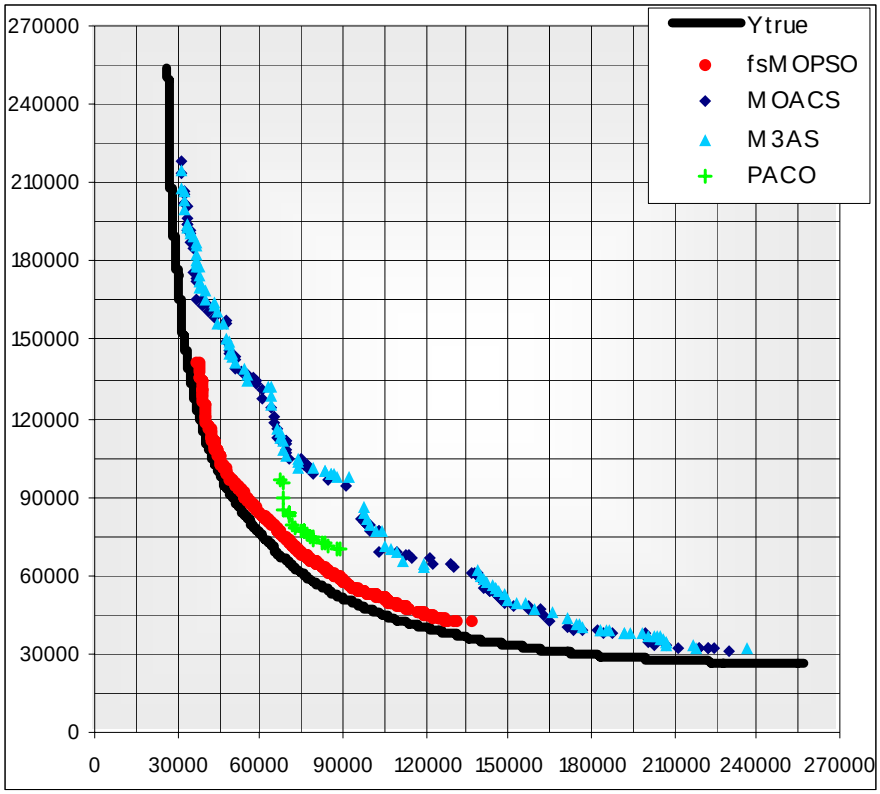


Figura 5.13. Mejores frentes Pareto aproximados del método fsMOPSO y de los métodos MOACOs para el problema KROAB150.

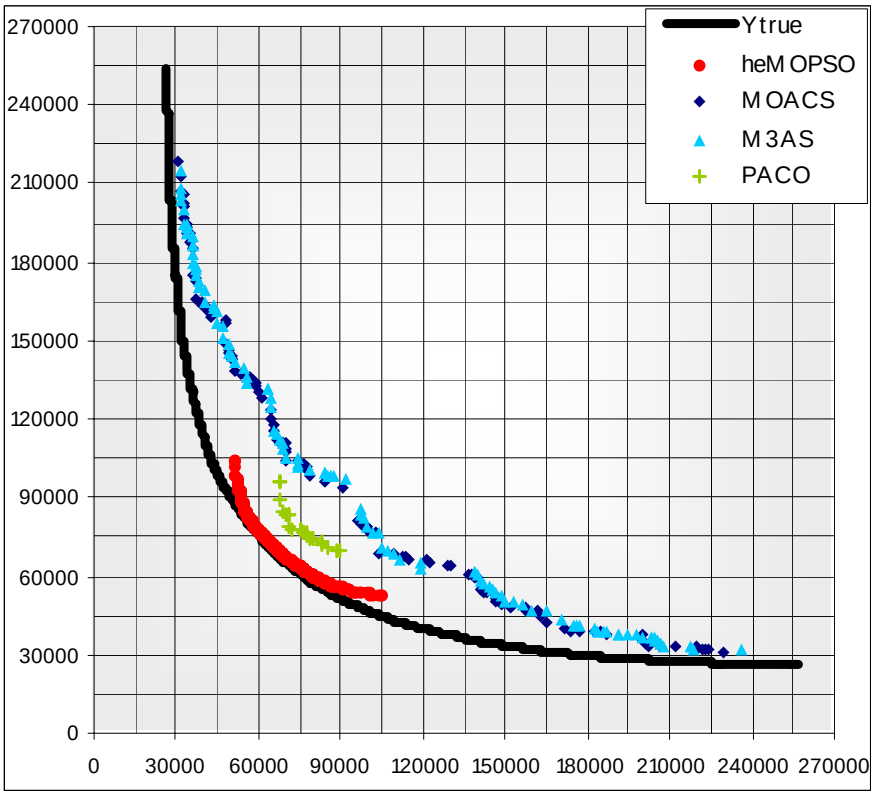


Figura 5.14. Mejores frentes Pareto aproximados del método heMOPSO y de los métodos MOACOs para el problema KROAB150.

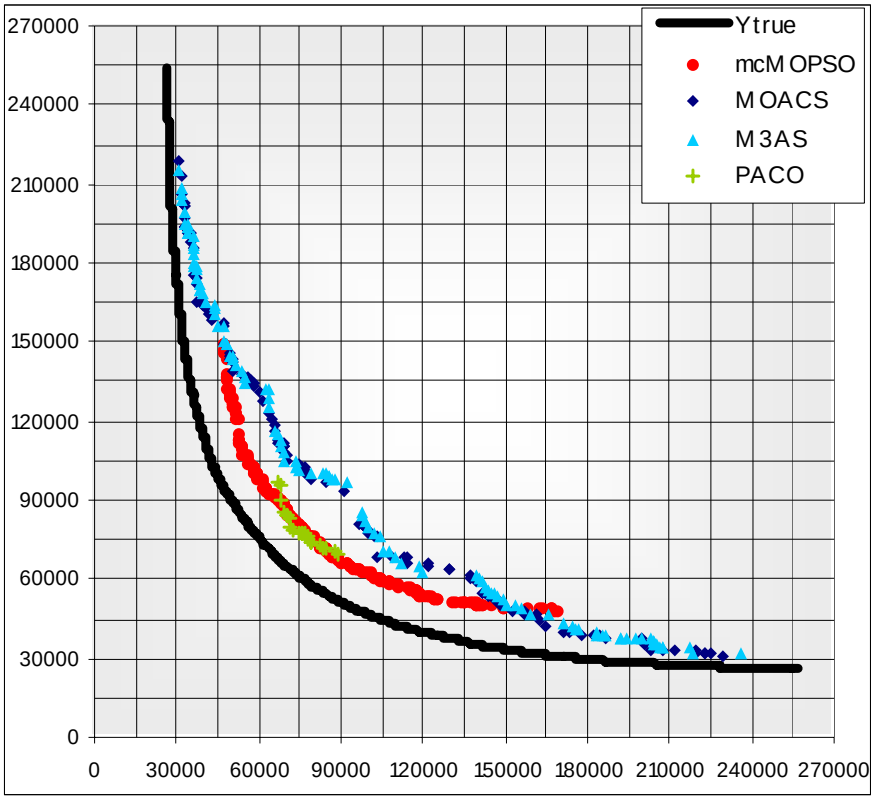


Figura 5.15. Mejores frentes Pareto aproximados del método mcMOPSO y de los métodos MOACOs para el problema KROAB150.

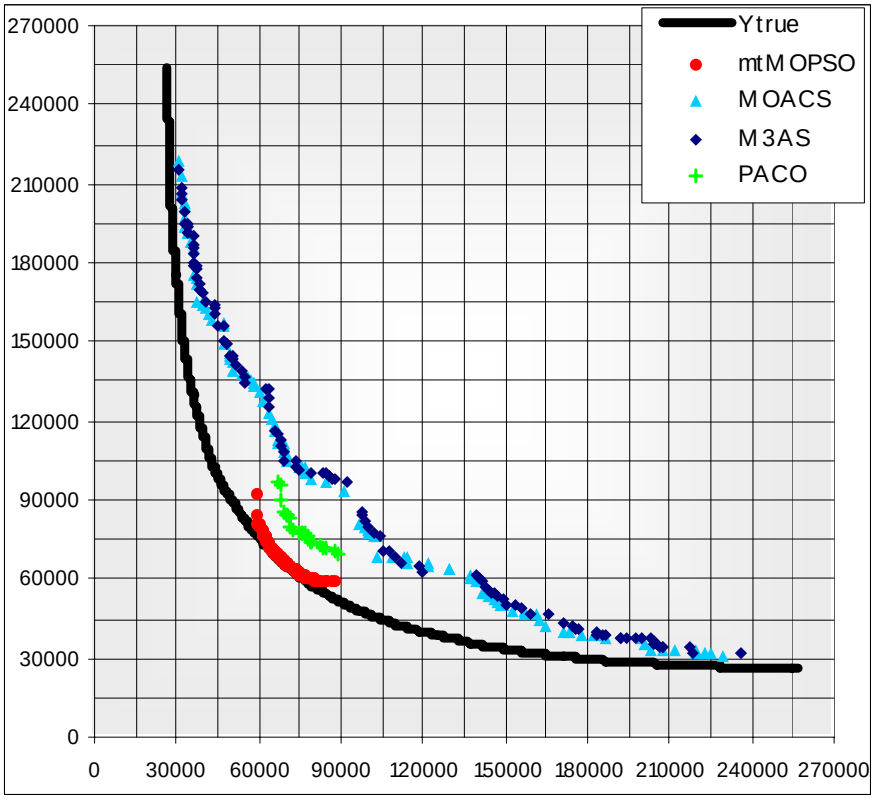


Figura 5.16. Mejores frentes Pareto aproximados del método mtMOPSO y de los métodos MOACOs para el problema KROAB150.

5.3.4 Problema KROAB200

Para el problema KROAB200 las evaluaciones normalizadas de cada corrida por cada método se presentan en las tablas 5.28 a 5.35, el ranking promedio en cada métrica se muestra en la tabla 5.36 y la figura 5.5 ilustra los mejores frentes Pareto aproximados por cada método MOPSO.

En este problema puede verse como las evaluaciones promedio en la métrica M_1^* presentan una disminución considerable en la mayoría de los métodos, con excepción de los métodos mtMOPSO y PACO, siendo más resaltante la reducción de la evaluación del método mcMOPSO.

Para la métrica M_2^* se observa que la mayoría de los métodos presentan una evaluación similar como en los casos anteriores, quedando nuevamente en los primeros lugares los métodos MOACOs.

En lo que respecta a la métrica M_3^*, es posible ver que los métodos M3AS y MOACS presentan una evaluación similar a la obtenida en los problemas anteriores, mientras que los demás métodos presentan una disminución considerable en su evaluación.

Por último, se observa que los métodos MOPSOs consiguen encontrar más soluciones que los métodos MOACOs.

Además, es posible observar en las gráficas que varios de los métodos MOPSOs son superados por el método PACO en términos de calidad de soluciones.

clMOPSO - KROAB200				
Corrida	M_1^*	M_2^*	M_3^*	**|Y'|**
1	0,35	0,80	0,64	1,00
2	0,27	0,80	0,63	0,89
3	0,38	0,80	0,64	0,98
4	0,34	0,80	0,62	0,86
5	0,32	0,80	0,60	0,87
6	0,36	0,81	0,63	0,97
7	0,28	0,80	0,65	0,96
8	0,31	0,79	0,64	0,88
9	0,26	0,80	0,70	0,93
10	0,29	0,80	0,60	0,84
Promedio	**0,31**	**0,80**	**0,63**	**0,92**
Desv. Est.	**0,04**	**0,01**	**0,03**	**0,06**

Tabla 5.28. Evaluaciones normalizadas de las corridas del método clMOPSO para el problema KROAB200.

fsMOPSO - KROAB200				
Corrida	M_1^*	M_2^*	M_3^*	**|Y'|**
1	0,40	0,80	0,57	0,76
2	0,36	0,81	0,55	0,92
3	0,34	0,81	0,56	0,90
4	0,31	0,80	0,56	0,89
5	0,43	0,79	0,54	0,87
6	0,40	0,81	0,56	0,66
7	0,35	0,80	0,56	0,78
8	0,34	0,82	0,57	0,79
9	0,29	0,81	0,60	0,83
10	0,45	0,77	0,57	0,73
Promedio	**0,37**	**0,80**	**0,56**	**0,81**
Desv. Est.	**0,05**	**0,01**	**0,01**	**0,08**

Tabla 5.29. Evaluaciones normalizadas de las corridas del método fsMOPSO para el problema KROAB200.

heMOPSO - KROAB200				
Corrida	M_1^*	M_2^*	M_3^*	**|Y'|**
1	0,75	0,76	0,37	0,77
2	0,58	0,80	0,41	0,61
3	0,72	0,80	0,39	0,67
4	0,69	0,77	0,34	0,68
5	0,62	0,78	0,40	0,60
6	0,81	0,75	0,37	0,56
7	0,74	0,75	0,38	0,67
8	0,75	0,78	0,36	0,63
9	0,71	0,81	0,38	0,57
10	0,65	0,79	0,36	0,52
Promedio	**0,70**	**0,78**	**0,38**	**0,63**
Desv. Est.	**0,07**	**0,02**	**0,02**	**0,07**

Tabla 5.30. Evaluaciones normalizadas de las corridas del método heMOPSO para el problema KROAB200.

mcMOPSO - KROAB200				
Corrida	M_1^*	M_2^*	M_3^*	**|Y'|**
1	0,00	0,81	0,59	0,73
2	0,04	0,82	0,58	0,82
3	0,05	0,80	0,61	0,79
4	0,08	0,79	0,62	0,79
5	0,03	0,80	0,60	0,71
6	0,01	0,80	0,61	0,77
7	0,04	0,80	0,59	0,69
8	0,08	0,81	0,58	0,76
9	0,10	0,79	0,56	0,75
10	0,04	0,82	0,58	0,74
Promedio	**0,05**	**0,80**	**0,59**	**0,75**
Desv. Est.	**0,03**	**0,01**	**0,02**	**0,04**

Tabla 5.31. Evaluaciones normalizadas de las corridas del método mcMOPSO para el problema KROAB200.

mtMOPSO - KROAB200				
Corrida	M_1^*	M_2^*	M_3^*	**|Y'|**
1	0,90	0,66	0,27	0,96
2	0,92	0,58	0,28	0,93
3	0,92	0,68	0,31	0,75
4	0,88	0,69	0,28	0,96
5	0,91	0,66	0,29	0,75
6	0,85	0,70	0,29	0,71
7	0,93	0,70	0,27	0,76
8	0,88	0,60	0,31	0,71
9	0,93	0,65	0,28	0,77
10	0,87	0,71	0,28	0,89
Promedio	**0,90**	**0,66**	**0,29**	**0,82**
Desv. Est.	**0,03**	**0,05**	**0,01**	**0,10**

Tabla 5.32. Evaluaciones normalizadas de las corridas del método mtMOPSO para el problema KROAB200.

MOACS - KROAB200				
Corrida	M_1^*	M_2^*	M_3^*	**|Y'|**
1	0,28	0,83	0,92	0,30
2	0,31	0,84	0,92	0,36
3	0,28	0,83	0,92	0,40
4	0,24	0,83	0,92	0,23
5	0,26	0,83	0,92	0,42
6	0,31	0,83	0,92	0,39
7	0,31	0,83	0,91	0,34
8	0,31	0,83	0,92	0,37
9	0,34	0,82	0,91	0,36
10	0,28	0,82	0,92	0,34
Promedio	**0,29**	**0,83**	**0,92**	**0,35**
Desv. Est.	**0,03**	**0,00**	**0,00**	**0,05**

Tabla 5.33. Evaluaciones normalizadas de las corridas del método MOACS para el problema KROAB200.

M3AS - KROAB200				
Corrida	M_1^*	M_2^*	M_3^*	**|Y'|**
1	0,26	0,85	0,92	0,34
2	0,26	0,84	0,92	0,35
3	0,29	0,84	0,93	0,23
4	0,26	0,83	0,92	0,41
5	0,25	0,84	0,93	0,32
6	0,26	0,84	0,92	0,38
7	0,26	0,83	0,92	0,41
8	0,25	0,84	0,92	0,37
9	0,27	0,83	0,92	0,42
10	0,25	0,84	0,92	0,35
Promedio	**0,26**	**0,84**	**0,92**	**0,36**
Desv. Est.	**0,01**	**0,01**	**0,00**	**0,06**

Tabla 5.34. Evaluaciones normalizadas de las corridas del método M3AS para el problema KROAB200.

PACO - KROAB200				
Corrida	M_1^*	M_2^*	M_3^*	**|Y'|**
1	0,31	0,83	0,26	0,04
2	0,33	0,80	0,26	0,06
3	0,34	0,85	0,26	0,05
4	0,34	0,85	0,26	0,05
5	0,36	0,88	0,27	0,06
6	0,36	0,89	0,27	0,07
7	0,36	0,80	0,27	0,07
8	0,37	0,79	0,27	0,07
9	0,37	0,81	0,28	0,08
10	0,38	0,78	0,28	0,08
Promedio	**0,35**	**0,83**	**0,27**	**0,06**
Desv. Est.	**0,02**	**0,04**	**0,01**	**0,01**

Tabla 5.35. Evaluaciones normalizadas de las corridas del método PACO para el problema KROAB200.

Ranking KROAB200								
Pos.	M_1^*		M_2^*		M_3^*		$\|Y'\|$	
1º	mtMOPSO	0,90	M3AS	0,84	M3AS	0,92	clMOPSO	0,92
2º	heMOPSO	0,70	MOACS	0,83	MOACS	0,92	mtMOPSO	0,82
3º	fsMOPSO	0,37	PACO	0,83	clMOPSO	0,63	fsMOPSO	0,81
4º	PACO	0,35	mcMOPSO	0,80	mcMOPSO	0,59	mcMOPSO	0,75
5º	clMOPSO	0,31	clMOPSO	0,80	fsMOPSO	0,56	heMOPSO	0,63
6º	MOACS	0,29	fsMOPSO	0,80	heMOPSO	0,38	M3AS	0,36
7º	M3AS	0,26	heMOPSO	0,78	mtMOPSO	0,29	MOACS	0,35
8º	mcMOPSO	0,05	mtMOPSO	0,66	PACO	0,27	PACO	0,06

Tabla 5.36. Ranking de métodos por cada métrica en el problema KROAB200.

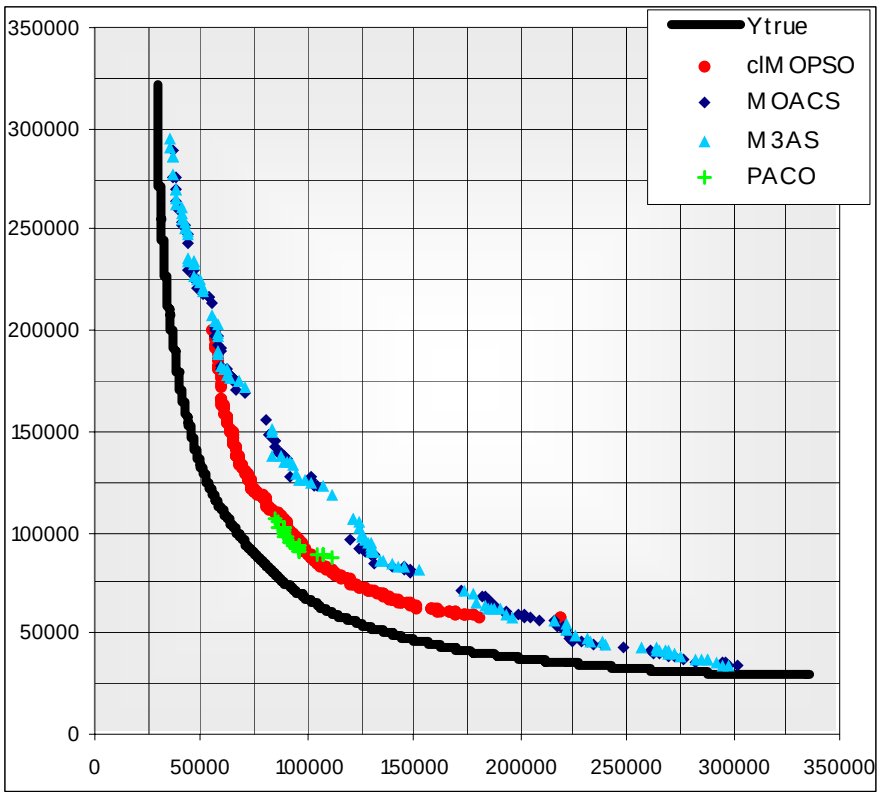


Figura 5.17. Mejores frentes Pareto aproximados del método clMOPSO y de los métodos MOACOs para el problema KROAB200.

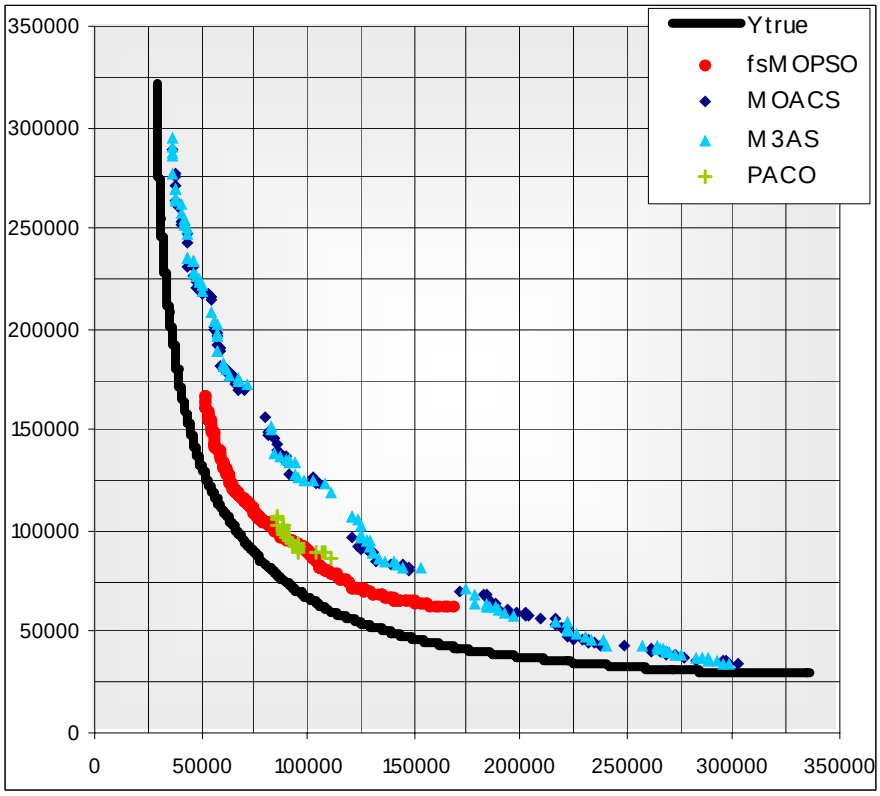


Figura 5.18. Mejores frentes Pareto aproximados del método fsMOPSO y de los métodos MOACOs para el problema KROAB200.

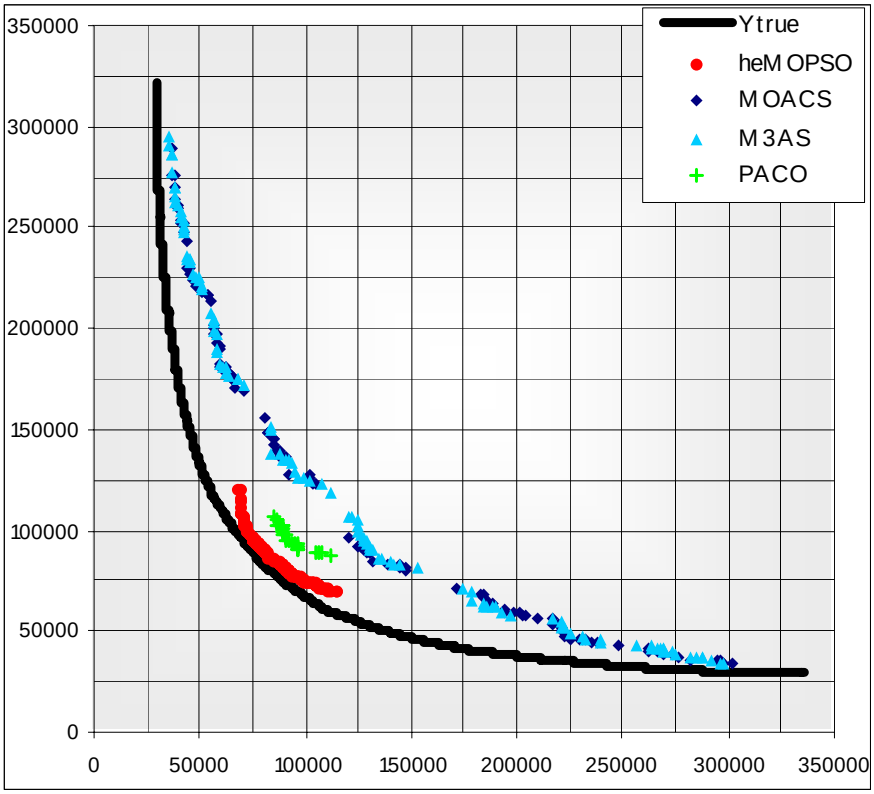


Figura 5.19. Mejores frentes Pareto aproximados del método heMOPSO y de los métodos MOACOs para el problema KROAB200.

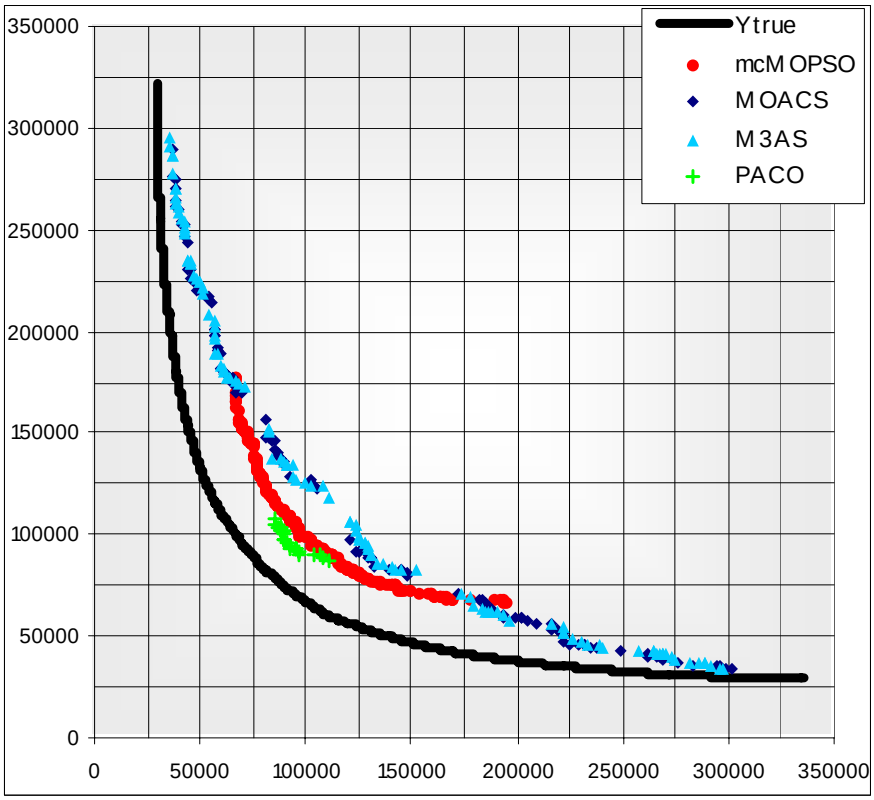


Figura 5.20. Mejores frentes Pareto aproximados del método mcMOPSO y de los métodos MOACOs para el problema KROAB200.

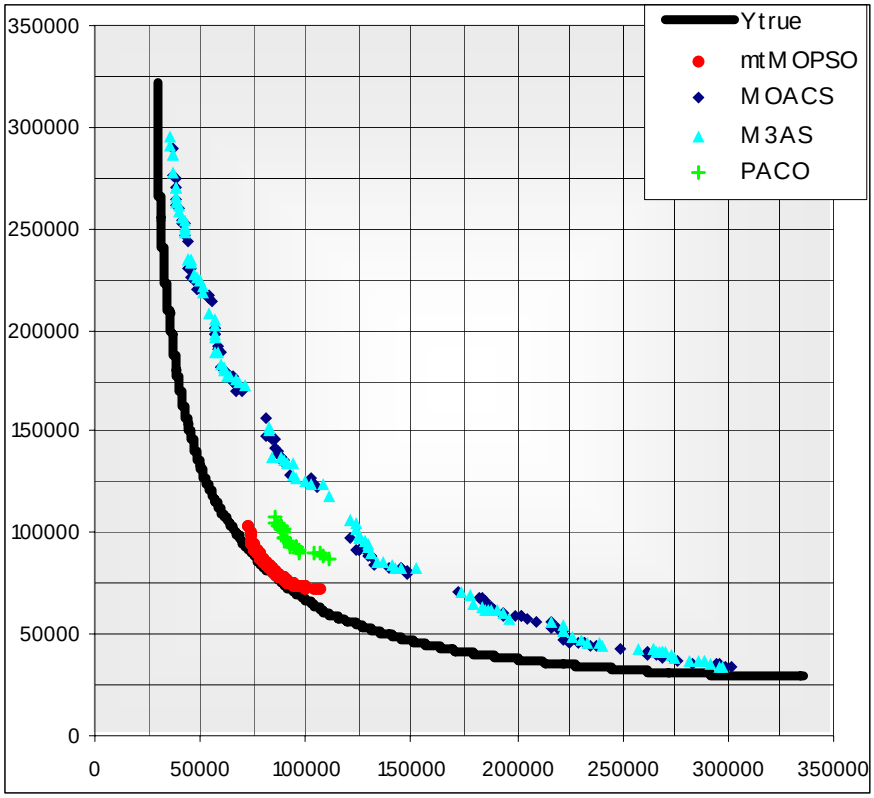


Figura 5.21. Mejores frentes Pareto aproximados del método mtMOPSO y de los métodos MOACOs para el problema KROAB200.

5.3.5 Rankings y Conclusiones Generales

En la tabla 5.37 se muestra el ranking de evaluaciones promedios en cada métrica considerando todos los problemas resueltos. Además, se presenta un *ranking promedio general* confeccionado a partir del promedio de las evaluaciones de cada método en todas las métricas y todos los problemas.

Pos.	M_1^*		M_2^*		M_3^*		$\|Y'\|$		**Ranking General**	
1º	mtMOPSO	0,90	M3AS	0,83	M3AS	0,93	clMOPSO	0,93	clMOPSO	0,78
2º	heMOPSO	0,83	PACO	0,82	MOACS	0,91	fsMOPSO	0,84	fsMOPSO	0,74
3º	fsMOPSO	0,67	MOACS	0,82	clMOPSO	0,79	heMOPSO	0,78	heMOPSO	0,71
4º	clMOPSO	0,60	clMOPSO	0,80	mcMOPSO	0,66	mtMOPSO	0,70	mtMOPSO	0,65
5º	PACO	0,35	mcMOPSO	0,80	fsMOPSO	0,65	mcMOPSO	0,58	mcMOPSO	0,58
6º	mcMOPSO	0,28	fsMOPSO	0,80	heMOPSO	0,47	MOACS	0,29	M3AS	0,55
7º	M3AS	0,19	heMOPSO	0,75	mtMOPSO	0,36	M3AS	0,26	MOACS	0,55
8º	MOACS	0,17	mtMOPSO	0,67	PACO	0,32	PACO	0,05	PACO	0,39

Tabla 5.37. Ranking de métodos en todos los problemas.

A partir de los resultados en la tabla 5.37 puede concluirse lo siguiente:

Para la métrica de aproximación al frente Pareto real M_1^*, casi todos los métodos MOPSOs resultan superiores a los métodos MOACOs. Esto puede observase en las graficas, en donde la mayoría de los métodos MOPSOs presentan soluciones más próximas al frente

Pareto real, siendo el método mtMOPSO superior en todos los casos. Además, aunque la mayoría los métodos MOPSOs han ido perdiendo calidad en sus soluciones al aumentar la complejidad de los problemas, estos han sido capaces de encontrar mejores soluciones que los métodos M3AS y MOACS en la región en donde se optimizan ambos objetivos con igual importancia.

Para la métrica de distribución de soluciones M_2^*, todos los métodos presentan resultados similares, a excepción del método mtMOPSO. Aunque, es posible observar en las gráficas que los métodos MOACOs presentan frentes Pareto aproximados en los que se observa un amontonamiento de soluciones en ciertas áreas y huecos en otras, aun ante el hecho de que los mejores frentes Pareto aproximados se han conformado por la unión de los frentes Pareto aproximados obtenidos en todas sus corridas en cada problema resuelto, a los cuales son muy similares.

En cuanto a la extensión del los frentes Pareto aproximados M_3^*, es posible ver como los métodos M3AS y MOACS logran aproximar casi completamente la extensión de los frentes Pareto reales, logrando encontrar soluciones muy próximas a los extremos de estos frentes y que no han sido encontradas (o aproximadas) por los demás métodos. También se observa que los frentes Pareto aproximados de los métodos MOPSOs van perdiendo extensión a medida que aumenta la complejidad de los problemas.

En lo que respecta a la métrica de cantidad de soluciones por corrida $|Y'|$, es posible ver que los métodos MOPSOs son capaces de encontrar mayor cantidad de soluciones que los métodos MOACOs en todos los problemas considerados, lo cual brinda al Tomador de Decisiones mayor diversidad de soluciones.

Para el ranking promedio general presentado en las últimas columnas de la tabla 5.37, puede verse que todos los métodos MOPSOs superan a los métodos MOACOs. Figurando en *primer lugar* el método clMOPSO, el cual ha logrado en promedio una buena aproximación al frente Pareto real de todos los problemas, superado levemente por el método PACO sólo en el problema KROAB200. También ha conseguido una distribución de soluciones muy similar a la de los métodos MOACOs y frentes Pareto aproximados de muy buena extensión, posicionándose siempre en el tercer lugar en los rankings de la métrica de extensión. Además, el método clMOPSO ha encontrado la mayor cantidad de soluciones no dominadas entre si en cada problema resuelto.

La segunda posición del ranking promedio general corresponde al método fsMOPSO, el cual ha logrado superar a todos los métodos MOACOs y al método clMOPSO en todos los

casos en cuanto a la métrica M_1^*, logrando también obtener frentes Pareto aproximados de distribución similar a la de los demás métodos y con buena extensión. Además, el método fsMOPSO ha sido el que ha proporcionado la segunda mayor cantidad de soluciones no dominadas entre si en casi todos los problemas resueltos.

Al igual que el método fsMOPSO, el método heMOPSO debe su tercera posición en el ranking promedio general a su buena evaluación en los rankings promedio de calidad y cantidad de soluciones en los problemas resueltos, consiguiendo en todos los casos frentes Pareto aproximados de extensión media y con distribución de soluciones levemente inferior a la mayoría de los demás métodos.

La cuarta posición del ranking general corresponde al método mtMOPSO, debido en gran parte a que este método ha presentado en todos los casos de pruebas los frentes Pareto aproximados más cercanos a los frentes Pareto reales, destacándose además por concentrarse únicamente en las soluciones que optimizan con igual importancia ambos objetivos como se puede verse en las gráficas y a la vez, esta es la razón por la cual ha obtenido baja evaluación en los rankings de la métrica de extensión. También, este método ha mostrado tener un comportamiento medio en cuanto a la cantidad de soluciones no dominadas presentadas, lo cual puede observarse al ver que este método ocupa siempre las posiciones centrales en los rankings de esta métrica.

El método mcMOPSO ocupa la quinta posición en el ranking general debido principalmente a que en todos los problemas de prueba ha encontrado soluciones de mejor calidad que las soluciones proporcionadas por los métodos MOACS y M3AS y que a la vez son sólo levemente superadas por algunas soluciones del método PACO en los problemas KROAB150 y KROAB200. Además, aunque este método ocupa la última posición en el ranking de la métrica M_1^* para el problema KROAB200, puede verse en la grafica de los mejores frentes Pareto aproximados de dicho problema que la mayoría de las soluciones de este método dominan a las soluciones de los métodos MOACS y M3AS. Así mismo, para los demás problemas es posible ver que el método mcMOPSO obtiene evaluaciones medias en los rankings de todas las métricas.

Aunque los métodos M3AS y MOACS han ocupado en todos los casos las primeras posiciones en los rankings promedio de las métricas M_2^* y M_3^*, estos a menudo han conseguido los últimos lugares en los rankings de la métrica M_1^*, debido principalmente a la mala calidad de sus soluciones al ir aproximando los frentes Pareto reales hacia el área en donde se optimizan ambos objetivos con igual importancia y presentando también en todos los casos una baja

evaluación en lo que respecta a la métrica de cantidad promedio de soluciones. Siendo estas las razones que los hacen inferiores a los métodos MOPSOs en promedio y por lo cual quedan posicionados detrás de estos métodos en el ranking general.

La razón por la cual el método PACO ha obtenido el último lugar en el ranking promedio general se debe a sus bajas evaluaciones promedio en la métricas M_1^*, M_3^* y $|Y'|$, ocupando siempre la última posición en los rankings de las últimas dos métricas. Aun así, este método fue capaz de encontrar mejores soluciones que los métodos MOACS y M3AS en la zona en que se optimizan ambos objetivos con igual importancia para todos los problemas resueltos.

Este trabajo recomienda para la resolución de MOPs (o MOCOPs), siempre que no se conozca un mejor método, a los métodos clMOPSO y fsMOPSO si las intenciones del Tomador de Decisiones son aproximar el frente Pareto real con buenas soluciones, extensión media y alta cantidad de soluciones. Mientras que, si sólo se desean soluciones que consideren únicamente la optimización de todos los objetivos con la misma importancia se recomienda al método mtMOPSO, pues es el que más se ha acercado en esta zona a los frentes Pareto reales de los problemas resueltos.

Finalmente, si bien para este trabajo solo se ha considerado la comparación de los frentes Pareto aproximados obtenidos por cada método luego de cierto tiempo fijo de ejecución, considerado suficiente para que todos los métodos converjan a muy buenos resultados, vale la pena mencionar que pudo observarse que en los primeros instantes ejecución tanto los métodos MOPSOs como los métodos MOACOs logran aproximarse a los frentes Pareto reales de los problemas resueltos casi con la misma distancia separación y llegado cierto momento, los métodos MOACOs se estancan en los frentes mostrados en la figuras anteriores, mientras que los métodos MOPSOs siguen mejorando las soluciones encontradas hasta finalizar su ejecución. Por esta razón, se propone como trabajo futuro la realización de otros experimentos en los cuales se consideren la comparación a intervalos temporales de las soluciones devueltas por cada método, de manera a determinar cuales son los métodos que más rápidamente logran obtener buenas aproximaciones a los frentes Pareto reales según las métricas utilizadas en este trabajo.

Como no se realizaron pruebas que demoren más de 200 segundos de ejecución, es difícil de describir que tipos de resultados se conseguirían con los métodos implementados en tiempos de ejecución mayores a los 200 segundos, aunque según los resultados presentados y las observaciones del párrafo anterior, se espera que los métodos MOPSOs sigan aventajando a los métodos MOACOs en cantidad y calidad de soluciones.

Capítulo 6

Conclusiones y Trabajos Futuros

6.1 Conclusiones Finales

En los capítulos anteriores se han presentado una revisión de los siguientes temas:

- Conceptos fundamentales acerca de la formulación matemática de MOPs, de modo a introducir el tratamiento de este tipo de problemas y de sus soluciones.
- Técnicas tradicionales y técnicas basadas en EA para la resolución de MOPs, presentando un resumen de algunas técnicas y métodos existentes para tratar este tipo de problemas.
- Formulación mono-objetiva y multiobjetiva del TSP y la presentación de algunas de sus aplicaciones prácticas, de manera a mostrar la importancia de este problema no solo como caso de prueba para la comparación de métodos, sino como caso práctico en situaciones reales.
- Conceptos fundamentales de la metaheurística ACO y los métodos basados en esta técnica para la resolución de MOCOPs, los cuales fueron implementados para resolver el TSP bi-objetivo y compararlos contra los métodos MOPSOs.
- Métodos basados en técnicas tradicionales de resolución de MOPs y del TSP mono objetivo para la resolución del TSP multiobjetivo, cuyas soluciones disponibles en Internet sirvieron de referencia para la comparación de los métodos implementados.
- Conceptos fundamentales de la metaheurística PSO y su implementación para la resolución de problemas mono-objetivos y multiobjetivos.
- Métodos basados en la metaheurística PSO para la resolución de MOPs, los cuales fueron implementados para resolver instancias de prueba del TSP bi-objetivo, con lo cual se busca demostrar empíricamente que los conceptos de esta metaheurística también pueden ser aplicados para la resolución de MOCOPs.
- La adaptación aplicada a los métodos MOPSOs para la resolución del TSP multiobjetivo, de manera a mostrar como los conceptos de la metaheurística PSO son aplicados en la resolución del TSP multiobjetivo.

Fueron realizadas implementaciones de 5 métodos MOPSOs y 3 métodos MOACOs para la resolución del TSP bi-objetivo. Con estas implementaciones se llevaron a cabo 10 corridas

de cada método en 4 instancias bi-objetivas del TSP, dos de 100 ciudades, una de 150 ciudades y una de 200 ciudades.

Los frentes Pareto aproximados en cada corrida para cada problema fueron comparados contra un conjunto de soluciones no dominadas entre si, obtenidas por los métodos PLS y PD-TPLS propuestos por Paquete et al. [Paquete03, Paquete04], utilizando para realizar las comparaciones las métricas M_1^*, M_2^* y M_3^* propuestas por Zitzler et al. [Zitzler00], que evalúan la calidad de las soluciones, la distribución de soluciones y la extensión del frente Pareto aproximado devuelto en cada corrida, y una métrica complementaria que considera la cantidad de soluciones de cada frente Pareto aproximado. De esta forma, las comparaciones tienen en cuenta características deseables en una buena aproximación al frente Pareto real de un MOP y son realizadas considerando las mejores aproximaciones conocidas de los frentes Pareto de los problemas de prueba.

Las evaluaciones de las 10 corridas de cada método fueron normalizadas a un número entre 0 y 1. Luego, se encontraron los valores medios de evaluación de cada método en cada métrica para cada problema resuelto. A partir de estos valores medios, se confeccionaron rankings de métodos en cada métrica para cada problema, rankings de métodos en cada métrica considerando todos los problemas y un ranking general de métodos que considera todos los resultados obtenidos.

Además, a partir de las soluciones encontradas se han confeccionado gráficas que ayudan a complementar los resultados mostrados en los rankings y que permiten tener una visión global de las soluciones obtenidas por cada método implementado, que al mismo tiempo son contrastadas por las soluciones obtenidas por los métodos PLS y PD-TPLS.

A partir de los rankings en cada métrica y las gráficas, se han presentado conclusiones parciales acerca del desenvolvimiento de los métodos MOPSOs y MOACOs en cada problema en particular. También, a partir del ranking general se han propuesto conclusiones parciales acerca de las posiciones que han obtenido los métodos en dicho ranking.

Finalmente, de las conclusiones parciales es posible derivar las siguientes conclusiones finales:

- Los métodos MOPSOs presentan mejores soluciones que los métodos MOACOs en términos de calidad y cantidad de soluciones, lo cual puede verse en la tablas 5.9, 5.18, 5.27, 5.36 y 5.37 y las figuras 5.2 a 5.4.
- Aunque los métodos MOPSOs no hayan obtenido los primeros lugares en los rankings de la métrica de distribución, puede observase en las gráficas que estos métodos pre-

sentan frentes Pareto aproximados más densos y con menos huecos comparados con los frentes Pareto aproximados por los métodos MOACOs, con lo cual se concluye que los métodos MOPSOs son tan buenos como los métodos MOACOs en este criterio de comparación.

- Los métodos M3AS y MOACS presentan las mejores evaluaciones en los rankings de la métrica de extensión, pero resultan ser los peores métodos si lo que se pretende es aproximar con calidad los frentes Pareto de los problemas resueltos, especialmente en la zona donde se optimizan todos los objetivos con igual importancia.
- El método PACO, a pesar de que no ha a presentado aproximaciones a los frentes Pareto reales de los problemas resueltos que posean buena extensión, es el método MOACO que logra aproximar con mejor calidad la zona en donde se optimizan con igual importancia ambos objetivos.
- Se consideran como primer y segundo mejor método, teniendo en cuenta todas las métricas utilizadas, a los métodos clMOPSO y fsMOPSO, pues son los que han obtenido las dos mejores evaluaciones en el ranking general de métodos.
- El método mtMOPSO se considera como el mejor método para aproximar la zona del frente Pareto real de un MOP en donde se optimizan con igual importancia todos los objetivos, pues este método ha proporcionado la mejor aproximación de esta zona de los frentes Pareto reales de los problemas resueltos.
- Aun al tratarse el TSP de un COP, para el cual en principio los métodos basados en la metaheurística PSO no son apropiados, estos lograron aplicarse eficientemente con la adaptación usada en este trabajo. Con esto se demuestra empíricamente que los conceptos de la metaheurística PSO también pueden ser aplicados con éxito en la resolución de COPs.
- La razón por la cual los métodos MOPSOs y MOACOs no logran conseguir mejores soluciones que los métodos PLS y PD-TPLS radica en que los últimos utilizan heurísticas específicas para el TSP, mientras que los primeros se basan en estrategias genéricas de búsqueda inspiradas en la naturaleza. Pero a la vez, se comprueba que las estrategias genéricas de los métodos MOPSOs y MOACOs son capaces de producir buenos resultados y es de esperarse que estos métodos resulten efectivos cuando las técnicas específicas y/o tradicionales no puedan ser aplicadas.

En resumen, los principales aportes de este trabajo son:

- La resolución efectiva de un COP a través de los conceptos de la metaheurística PSO, lo cual demuestra empíricamente la validez de la aplicación de los conceptos de esta metaheurística en este tipo de problemas.
- La resolución efectiva de un MOP a través de MOPSOs, mostrando que estos métodos son tan buenos como otros métodos de resolución de este tipo de problemas.
- La comparación de dos técnicas de resolución alternativa a través de algunos métodos representativos de sus estados del arte y mediante resultados obtenidos por métodos basados en técnicas tradicionales.

6.2 Trabajos Futuros

De manera a que se pueda continuar con el trabajo iniciado en esta tesis de grado, los siguientes puntos son propuestos como trabajos futuros:

- Considerar la aplicación de los métodos MOPSOs a otros COPs, tales como el *Vehicle Routing Problem* (VRP) [Barán03], el *Quadratic Assignment Problem* (QAP) [Paciello06] y el enrutamiento de datos en redes de computadores [Pinto05].
- Combinar los métodos MOPSOs, MOACOs, MOGLS, PLS y PD-TPLS en un *Equipo Paralelo de Algoritmos* (*Team Algorithms*) [Paciello06] para la resolución de problemas TSP multiobjetivos de mayor número de ciudades y de otros MOPs con espacios de búsqueda complejos y de alta dimensionalidad.
- Incorporar otros métodos MOPSOs en el estudio realizado en este trabajo.
- Idear algún esquema de repulsión entre las partículas para mejorar la habilidad de exploración de los métodos MOPSOs e incrementar la extensión de los frentes Pareto aproximados devueltos por dichos métodos.
- Encontrar casos prácticos relacionados a problemas reales en los que puedan ser aplicados los métodos implementados en este trabajo.
- Realizar otros experimentos en los cuales se consideren la comparación de las soluciones devueltas por cada método en intervalos temporales, de manera a determinar cuales son los métodos que más rápidamente logran obtener buenas aproximaciones a los frentes Pareto reales según las métricas utilizadas en este trabajo.

Referencias Bibliográficas

[Agarwala00] R. Agarwala, D. L. Applegate, D. Maglott, G. D. Schuler, A. A. Schiffer. A fast and scalable radiation hybrid map construction and integration strategy. Genome Research 10, 350-364. 2000.

[Bailey00] C. A. Bailey, T. W. McLain, and R. W. Beard. Fuel saving strategies for separated spacecraft interferometry. In AIAA Guidance, Navigation and Control Conference. 2000.

[Barán03] B. Barán and M. Schaerer. A multiobjective Ant Colony System for Vehicle Routing Problems with Time Windows. Proc. Twenty first IASTED International Conference on Applied Informatics. Insbruck, Austria, pg. 97-102. 2003.

[Beyer02] H. Beyer and H. Schwefel. Evolution Strategies, A comprehensive introduction. Natural Computing 1: 3-52. Kluver Academic Publishers. 2002

[Boese94] K. D. Boese, A. B. Kahng and R. S. Tsay. Scan Chain Optimization: Heuristic and Optimal Solutions. Internal Report, UCLA CS Dept., 1994.

[Coello99] C. Coello. An updated Survey of Evolutionary Multiobjective Optimization Techniques: state of the art and future trends. In Congress on Evolutionary Computation. Piscataway, N. J., IEEE Service Center. 3–13. 1999.

[Coello02] C. A. Coello Coello and M. S. Lechuga. MOPSO: A Proposal for Multiple Objective Particle Swarm Optimizations. In Congress on Evolutionary Computation (CEC'2002), volume 2, pages 1051–1056, Piscataway, New Jersey. 2002.

[Cramer85] N. L. Cramer. A representation for the adaptive generation of simple sequential programs. In J.J. Grefenstette, editor, Proceedings of an International Conference on Genetic Algorithms and the Applications, pages 183–187, Pittsburgh, PA., 1985. Carnegie-Mellon University.

[Deb02] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA–II. IEEE Transactions on Evolutionary Computation, vol. 6, no. 2, pp 182-197. 2002.

[Doerner02] K. Doerner et al. Pareto Ant Colony Optimization: A Metaheuristic Approach to Multiobjective Portfolio Selection, Proceedings of the 4th. Metaheuristics International Conference. Porto, 243-248. 2002.

[Dorigo96] M. Dorigo et al. The Ant System: Optimization by a colony of cooperating agents, IEEE Transactions on Systems, Man, and Cybernetics - Part B, 26, 1, 29-41. 1996.

[Dorigo97] M. Dorigo and L. Gambardella. Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem, IEEE Transactions on Evolutionary Computation, 1:1, 53-66. 1997.

[Eberhart95] R. C. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. Proceedings of the Sixth International Symposium on Micromachine and Human Science, Nagoya, Japan. pp. 39-43, 1995

[Eberhart98] Eberhart, R. C. and Shi, Y. Evolving artificial neural networks. Proceedings of International Conference on Neural Networks and Brain, 1998, Beijing, P. R. China. pp. PL5-PL13, 1998.

[Eberhart99] R. Eberhart, and X. Hu. Human tremor analysis using particle swarm optimization. Proceedings of Congress on Evolutionary Computation, 1999. Washington D.C., pp. 1927-1930, 1999.

[Fieldsend01] J. E. Fieldsend and S. Singh. Using Unconstrained Elite Archives for Multi Objective Optimization, IEEE Transactions on Evolutionary Computation (Submitted). 2001.

[Fieldsend02] J. E. Fieldsend and S. Singh. A Multi-Objective Algorithm based upon Particle Swarm Optimization, an Efficient Data Structure and Turbulence. In Proceedings of the 2002 U.K. Workshop on Computational Intelligence, pages 37-44, Birmingham, UK. 2002.

[Fogel64] L. J. Fogel. On the organization of intellect. PhD thesis, University of California, Los Angeles, California, 1964.

[Fukuyama01] Y. Fukuyama and H. Yoshida. A particle swarm optimization for reactive power and voltage control in electric power systems. Proceedigns of the IEEE Congress on Evolutionary Computation (CEC 2001), Seoul, Korea. 2001.

[Gómez04] O. Gómez and B. Barán. Omicron ACO, Proceedings of CLEI'2004. Latin-American Conference on Informatics (CLEI). Arequipa, Perú. 2004.

[Gupta03] P. Gupta, A. B. Kahng, S. Mantik. Routing-Aware Scan Chain Ordering. Proceedings of Asia and. South Pacific Design Automation Conf., pp. 857-862. 2003.

[Hansen98] M.P. Hansen. Metaheuristics for multiple objective combinatorial optimization. PhD thesis, Institute of Mathematical Modelling, Technical University of Denmark, Lyngby, Denmark, 1998.

[Heppner90] F. Heppner and U. Grenander. A stochastic nonlinear model for coordinated bird flocks. In S. Krasner, Ed., The Ubiquity of Chaos. AAAS Publications, Washington, DC. 1990.

[Holland92] J. H. Holland. Adaptation in Natural an Artificial Systems. MIT Press, Cambridge, Massachusetts, second edition, 1992.

[Horn93] 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.

[Horn94] 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.

[Hu02] X. Hu and R. Eberhart. Multiobjective Optimization Using Dynamic Neighborhood Particle Swarm Optimization. In Congress on Evolutionary Computation (CEC'2002), volume 2, pages 1677-1681, Piscataway, New Jersey. 2002.

[Hu03] X. Hu, R. Eberhart, and Y. Shi. Engineering optimization with particle swarm, IEEE Swarm Intelligence Symposium 2003, Indianapolis, IN, USA, 2003.

[Jaszkiewicz02] A. Jaszkiewicz. Genetic local search for multiple objective combinatorial optimization, European Journal of Operational Research, Vol. 137, No. 1, pp. 50--71, 2002.

[Kennedy95] J. Kennedy and R. Eberhart. Particle Swarm Optimization, In Proceedings of the 1995 IEEE International Conference on Neural Networks, pages 1942–1948, Piscataway, New Jersey. 1995.

[Koza89] J. R. Koza. Herarchical genetic algorithms operating on populations of computer programs. In N. S. Sridharan, editor, Proceedings of 11th International Joint Conference on Artificial Intelligence, pages 768–774, San Mateo, California, 1989. Morgan Kaufmann.

[Knowles00] J. Knowles and D. Corne. Approximating the nondominated front using the Pareto Archived Evolution Strategy. Evolutionary Computation, Vol. 8, no. 2, pp 149-172. 2000

[Lara03] A. Lara López. Un estudio de las Estrategias Evolutivas para Problemas Multiobjetivos. Tesis de Maestría en Ciencias. Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional. México. 2003.

[Lipschutz93] S. Lipschutz. Teoría y Problemas de Matemática Discreta. Serie de Compendios Schaum. Editorial McGraw Hill. 1993.

[Lücken03] C. Von Lücken. Algoritmos evolutivos para optimización multiobjetivo: Un estudio comparativo en un ambiente paralelo asíncrono. Tesis de Maestría en Ingeniería de Sistemas. Universidad Nacional de Asunción. 2003

[Metelco] High School Operations Research. Case Studies: "Metelco S.A. Efficient Drilling of Printed Circuit Boards". http://www.hsor.org/case_studies.cfm?name=metelco_greece.

[Moore99] J. Moore y R. Chapman. Application of Particle Swarm to Multiobjective Optimization. Department of Computer Science and Software Engineering, Auburn University, 1999.

[Mostaghim03] S. Mostaghim and J. Teich. Strategies for Finding Good Local Guides in Multiobjective Particle Swarm Optimization (MOPSO). In 2003 IEEE Swarm Intelligence Symposium Proceedings, pages 26–33, Indianapolis, Indiana, USA. 2003.

[Nilsson03] C. Nilsson. Heuristics for the Traveling Salesman Problem. 2003. www.ida.liu.se/~TDDB19/reports_2003/htsp.pdf.

[Paciello06] J. Paciello, H. Martínez, C. Lezcano and B. Barán. Algoritmos de Optimización multi-objetivos basados en colonias de hormigas. Proceedings of CLEI'2006. Latin-American Conference on Informatics (CLEI). Santiago, Chile. 2000.

[Paquete03] L. Paquete and T. Stützle. A two-phase local search for the biobjective traveling salesman problem. In EMO 2003, LNCS 2632, pages 479-493. Springer Verlag, 2003.

[Paquete04] L. Paquete, M. Chiarandini, and T. Stützle. Pareto local optimum sets in the biobjective traveling salesman problem: An experimental study. In Metaheuristics for Multiobjective Optimisation, LNEMS 535. Springer Verlag, 2004.

[Parsopoulos02] K. Parsopoulos and M. Vrahatis. Particle Swarm Optimization in Multiobjective Problems. In Proceedings of the 2002 ACM Symposium on Applied Computing (SAC'2002), pages 603-607, Madrid, Spain, 2002.

[Pinto05] D. Pinto and B. Barán. Solving Multiobjective Multicast Routing Problem with a new Ant Colony Optimization approach. LANC'05, Cali, Colombia. 2005. [Sanhi76] S. Sahni and T. Gonzalez. P-complete approximation problems. Journal of the ACM, 23:555-565, 1976.

[Reynolds87] C. W. Reynolds. Flocks, herds and schools: a distributed behavioral model. Computer Graphics, 21(4):25-34. 1987.

[Secrest01] B. Secrest. Traveling Salesman Problem For Surveillance Mission Using Particle Swarm Optimization. MS Thesis, AFIT/GCE/ENG/01M-03, School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, Ohio. 2001.

[Shi98a] Y. Shi, and R. C. Eberhart, A modified Particle Swarm Optimizer, IEEE International Conference on Evolutionary Computation, Anchorage, Alaska, 1998.

[Shi98b] Y. Shi y R. Eberhart. Parameter Selection in Particle Swarm Optimization. In Proceedings of the Seventh Annual Conference on Evolutionary Programming, pg. 591-601, 1998.

[Srinivas94] N. Srinivas and Kalyanmoy Deb. Multiobjective Optimization Using Non-dominated Sorting in Genetic Algorithms. Evolutionary Computation, 2(3):221–248, 1994.

[Stutzle00] T. Stutzle and H. Hoos. Max-Min Ant System. Future Generation Computer Systems, 16:8, 889-914. 2000.

[Toscano01] G. Toscano. Optimización Multiobjetivo Usando Un Micro Algoritmo Genético. Tesis de Maestría. Universidad Veracruzana – LANIA. 2001.

[Veldhuizen99] 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. 1999.

[Vesterstrom02] J. Vesterstrom and J. Riget. Particle Swarms. Masters Thesis. University of Aarhus. 2002.

[Wikipedia1] Combinatorial Optimization. Wikipedia. http://en.wikipedia.org/wiki/Combinatorial_optimization

[Wikipedia2] Danza de la Abeja. Wikipedia. http://es.wikipedia.org/wiki/Danza_de_la_abeja

[Yu98] G. Yu. Industrial Applications of Combinatorial Optimization, Kluwer Academic Publisher, Boston, 1998.

[Zitzler98] 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, 1998.

[Zitzler00] E. Zitzler, K. Deb and L. Thiele. Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. Evolutionary Computation, vol. 8, no.2, pp 173–195. 2000.

[Zitzler02] E. Zitzler, M. Laumanns and L. Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization. In K.C. Giannakoglou et al., editor, Proceedings of the EUROGEN2001 Conference, pages 95–100, Barcelona, Spain. 2002.