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

Algoritmos Culturales Aplicados a Optimización con Restricciones y Optimización Multiobjetivo

Tesis que presenta

Ricardo Landa Becerra

Para obtener el grado de

Maestro en Ciencias

En la especialidad de

Ingeniería Eléctrica
Opción Computación

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

México, D. F., diciembre de 2002

Agradecimientos

A todos mis seres queridos, por apoyarme y estar conmigo. Les dedico esta tesis a ustedes.

Al Dr. Coello, por su acertada dirección y por compartir sus conocimientos.

Al CONACyT por la beca proporcionada durante la primera parte de mi estancia en el programa de maestría. Se agradece también la beca terminal de maestría proporcionada a través del proyecto "Estudio y Desarrollo de Técnicas Avanzadas de Manejo de Restricciones para Algoritmos Evolutivos en el Contexto de Optimización Numérica" (Ref. 32999-A), cuyo responsable es el Dr. Carlos A. Coello Coello.

Índice General

Índice de Figuras

Índice de Tablas

Índice de Algoritmos

Resumen

En esta tesis se proponen dos algoritmos culturales: uno para optimización mono-objetivo con restricciones y otro para optimización multiobjetivo. Ambos algoritmos están basados en la programación evolutiva. Estos algoritmos, por ser evolutivos, son capaces de operar con cualquier tipo de función objetivo.

Los algoritmos culturales son técnicas de computación evolutiva que operan en dos espacios: el espacio de la población (común entre los algoritmos evolutivos), y el espacio de creencias, en el que se almacenan experiencias (positivas o negativas) que la población ha adquirido a lo largo del proceso de búsqueda. Esta información sirve para guiar el proceso de búsqueda y generar nuevos individuos. La información puede influir en cualquiera de los operadores evolutivos.

En la técnica para optimización con restricciones, se utiliza el espacio de creencias para generar una especie de mapa del espacio de búsqueda, en el que se distinguen las regiones factible y no factible. La información se obtiene de los individuos generados con anterioridad, y se utiliza para generar nuevos individuos dentro de la región factible, y de preferencia en la frontera de la región factible y la región no factible.

Tal mapa se encuentra dentro de una ventana que cambia de ubicación para enfocarse en las zonas prometedoras. Para almacenarlo se utilizan 2^m-*trees*, que son estructuras de datos espaciales que ayudan a administrar la memoria del sistema de una forma más eficiente, al compararlas con estructuras de datos estáticas.

Los resultados de esta técnica se muestran competitivos frente a otros algoritmos evolutivos, requiriendo una menor cantidad de evaluaciones de la función objetivo.

Por otro lado, la técnica para optimización multiobjetivo utiliza un espacio de creencias, que también es una especie de mapa, pero en este caso en el espacio fenotípico (el espacio de las funciones objetivo). Con este espacio de creencias se tratan de identificar las zonas menos pobladas del frente de Pareto, y la influencia se ejerce sobre la selección, para conservar a los individuos que conformen un frente de Pareto mejor distribuido.

Los resultados de esta técnica muestran la factibilidad de un algoritmo cultural para optimización multiobjetivo. Cabe agregar que éste es el primer intento registrado de extender un algoritmo cultural a optimización multiobjetivo.

Abstract

Two cultural algorithms are presented in this thesis: a cultural algorithm for constrained single-objective optimization, and another one for multiobjective optimization. Both algorithms are based on evolutionary programming. Being evolutionary algorithms, these techniques are able to work with any type of objective function.

Cultural algorithms are evolutionary computation techniques which operate in two spaces: the population space (common among evolutionary algorithms), and the belief space, in which the population experiences, acquired along the search process, are stored. This information is used to guide the search process and to generate new individuals. The information can influence any evolutionary operator.

In the constrained optimization approach, the belief space is used to generate a feasibility map of the search space. The information is obtained from the individuals previously generated, and it is used to generate new individuals inside the feasible region, and if possible, in the boundary between the feasible and the infeasible region.

Such a map is inside a slicing window as to focus the search effort into promising regions. 2^m-trees are used to store the map. 2^m-trees are spatial data structures that are able to manage the system's memory better than a static data structure.

The results produced by this approach, when compared to other evolutionary algorithms, are competitive, and require a smaller number of fitness function evaluations.

On the other hand, the multiobjective optimization approach uses a belief space that is also a type of map, but in this case it is built in phenotypical space (objective function space). This belief space is used to identify regions of the Pareto front with less individuals, and it influences the selection to preserve individuals that form a better distributed Pareto front.

The results of this approach show the feasibility of a cultural algorithm for multiobjective optimization. It is worth adding that this is the first recorded attempt to extend a cultural algorithm for multiobjective optimization.

Introducción

Los problemas de optimización han sido sumamente estudiados, debido seguramente a la importancia que tienen en la práctica. Algunos problemas han logrado resolverse satisfactoriamente mediante métodos matemáticos, como la optimización lineal. Pero los problemas no lineales, en su caso general, no pueden resolverse por método determinístico alguno en un tiempo polinomial. Por esa razón, las heurísticas han tomando un gran auge, y entre ellas, los algoritmos de computación evolutiva.

En esta tesis se atacan los problemas de optimización global (mono-objetivo y con restricciones), y los problemas de optimización multiobjetivo. Para resolver ambos tipos de problemas se utilizan algoritmos culturales, que son un tipo especial de técnicas evolutivas en las cuales se tiene un "espacio de creencias" que ayuda a guiar la búsqueda, además de los operadores del algoritmo evolutivo que se emplee.

En el capítulo 1 se proporcionan los conceptos básicos acerca del tipo de problemas que se abordan en esta tesis, así como un breve repaso a los distintos paradigmas dentro de la computación evolutiva. Luego se habla brevemente de las formas en las que se ha incorporado conocimiento del dominio en los algoritmos evolutivos, para pasar a los algoritmos culturales.

En el capítulo 2 se habla acerca del trabajo previo en algoritmos evolutivos para optimización con restricciones, desde las funciones de penalización, hasta los algoritmos culturales, pasando por diversas técnicas. Dentro de los algoritmos culturales que se han usado para optimización con restricciones, se distinguen los llamados "CAEP" (*Cultural Algorithms with Evolutionary Programming*, Algoritmos Culturales con Programación Evolutiva), cuyo planteamiento es importante, debido a que los algoritmos propuestos en esta tesis también son CAEP. En este capítulo se describe la estructura de los CAEP.

En el capítulo 3 se describe la técnica propuesta para manejo de restricciones, comenzando con la estructura general del algoritmo. Se presta especial atención al espacio de creencias en esta descripción, puesto que ahí radica la propuesta del algoritmo, y se explican con detalle las diferencias con otras técnicas. Se habla también de la estructura de datos que se utilizó para hacer más eficiente el almacenamiento del espacio de creencias. Por último, se describen los pasos restantes del algoritmo, y se muestran los parámetros que requiere, así como algunos lineamientos para su asignación.

En el capítulo 4 se proporcionan los resultados de la técnica para manejo de restricciones, comenzando con las funciones de prueba utilizadas. Más adelante, se menciona la forma en la que se obtuvieron los resultados, y se comparan contra dos técnicas importantes: la jerarquización estocástica y los mapas homomorfos.

En el capítulo 5 se mencionan, primeramente, algunos conceptos de optimización multiobjetivo. Luego se habla acerca del trabajo previo en optimización evolutiva multiobjetivo. Las técnicas se dividen según el momento en el que se incorporan las preferencias de los objetivos. En este capítulo no se habla de algoritmos culturales puesto que no se tiene información de ninguna propuesta previa.

En el capítulo 6 se describe la propuesta para optimización multiobjetivo, siguiendo una estructura muy similar a la del capítulo 4, es decir, primero se habla de la estructura general del algoritmo, luego se describe el espacio de creencias utilizado, y posteriormente se detallan los pasos restantes del algoritmo. Al final están los parámetros que requiere la técnica, y algunas sugerencias para asignarlos.

En el capítulo 7 están los resultados de este algoritmo para optimización multiobjetivo. Primero se muestran las funciones de prueba utilizadas. Posteriormente se indica cómo fueron realizados los experimentos, y se muestran algunas gráficas con los resultados obtenidos. Finalmente, se aplicaron algunas métricas para evaluar los resultados, que son comparados contra un algoritmo representativo del estado del arte en optimización evolutiva multiobjetivo, el NSGA-II.

Capítulo 1

Conceptos Básicos

1.1 Optimización

La optimización es una de las áreas de las matemáticas que más aplicaciones tiene en la vida real. En procesos industriales aparecen constantemente problemas de optimización, y en nuestra vida diaria los resolvemos a cada momento, aunque solemos hacerlo de manera inconsciente.

Estos problemas se describen mediante un conjunto de elementos, de los cuales, los más importantes son:

- Las *variables de decisión*. Son los valores que se modifican para resolver el problema.
- Las *funciones objetivo*. Es necesaria al menos una. Se expresan en términos de las variables de decisión, y el resultado de su evaluación es el que se desea optimizar (maximizar o minimizar).
- Las *restricciones*. Desigualdades o ecuaciones que se tienen que cumplir para que la solución se considere *factible*. Puede ocurrir que el problema no presente restricciones, en cuyo caso todas las soluciones son válidas, y el proceso de búsqueda se enfoca entonces en optimizar las funciones objetivo.

Se puede expresar el problema de forma matemática como sigue: encontrar un vector de solución ($\vec{x}$) que optimice:

$$f\left(\vec{x}\right)$$

sujeto a:

$$g_i(\vec{x}) \leq 0,\ i = 1, \ldots, n$$
$$h_j(\vec{x}) = 0,\ j = 1, \ldots, p$$

donde $\vec{x}$ es el vector solución, f es la función objetivo, g_i es el conjunto de n restricciones de desigualdad, y h_j es el conjunto de p restricciones de igualdad.

Los primeros problemas de optimización que se estudiaron y lograron resolverse, son los llamados de programación lineal, en los cuales todas las funciones son combinaciones lineales de las variables de decisión del problema. En la actualidad existen diversos métodos matemáticos que aseguran encontrar el óptimo en estos problemas, como por ejemplo, el método Simplex.

También existen algunos métodos para resolver problemas de optimización no lineal, pero requieren información adicional del problema: por ejemplo, la mayoría requieren la primera derivada de la función objetivo. Esta información no siempre está disponible en los problemas que se desean resolver en el mundo real (por ejemplo, no siempre la función objetivo es diferenciable).

Para el caso general, no se conoce ningún algoritmo que asegure encontrar el óptimo en un tiempo polinomial. Para obtener soluciones a estos problemas en tiempos más cortos, y de esta manera, hacerlas útiles en la práctica, lo único que puede hacerse es desarrollar heurísticas, tratando de que se comporten lo mejor posible [115].

En este punto se ha enfocado principalmente la investigación reciente en optimización (no lineal y combinatoria), que con la ayuda de las computadoras ha adquirido una dimensión nueva. Por la cualidad de las computadoras electrónicas de poder realizar cálculos numéricos a velocidades cada vez mayores, se pueden idear algoritmos que realicen una cantidad muy alta de operaciones, y hacerlos ejecutar por un número elevado de iteraciones. Todo esto, con el objetivo de mejorar las soluciones obtenidas en pasos anteriores.

Muchos paradigmas se han desarrollado dentro de las heurísticas, cada uno de ellos con diversas variantes, y que resuelven distintos tipos de problemas. Como ejemplos sobresalientes podemos mencionar el recocido simulado [76] y la búsqueda tabú [53].

Una de estas heurísticas que ha demostrado ser muy competitiva, obteniendo en algunos casos resultados mejores que los de cualquier otra

heurística utilizada con anterioridad, es la *computación evolutiva*.

1.2 Computación Evolutiva

La computación evolutiva está basada en las ideas de la selección natural, y su aplicación en un ambiente artificial [46, 54]. La selección natural es vista como un proceso de optimización [8], en el que paulatinamente los individuos de la población se van mejorando para adaptarse a su medio.

Para la computación evolutiva, un individuo es una solución potencial a un problema, codificada de acuerdo con el funcionamiento del algoritmo; y el medio donde se desenvuelve lo componen la función objetivo y las restricciones, las que nos dirán qué tan apto es el individuo para sobrevivir.

La computación evolutiva involucra métodos que son poblacionales, lo que significa que trabajan con varias soluciones a la vez, y no con una, como lo hacen la mayoría de las otras heurísticas (por ejemplo, el recocido simulado), con lo que evitan quedar atrapados en óptimos locales.

Durante la ejecución de un algoritmo evolutivo, a la población se le aplican algunos operadores probabilísticos, con lo que se obtienen nuevas soluciones, las cuales se conservan o se descartan mediante un mecanismo de selección. Este proceso se repite por un número determinado de iteraciones, que en el contexto de computación evolutiva se denominan generaciones. El número de generaciones puede ser dado por el usuario o definido por el propio algoritmo.

Los operadores probabilísticos más comunes son la recombinación y la mutación. En la recombinación se hace una mezcla de dos o más individuos, llamados padres, para obtener uno o más individuos nuevos, llamados hijos. La mutación es una alteración aleatoria sobre un individuo.

Las características mencionadas son comunes a los tres paradigmas principales de la computación evolutiva. Tales paradigmas son:

- Programación evolutiva.

  Fue propuesta por Lawrence J. Fogel [47], con el principal objetivo de desarrollar máquinas de estados finitos. Posteriormente, David Fogel describió el algoritmo de programación evolutiva para optimización numérica [45].

Algoritmo 1 Programación evolutiva

```
Generar la población inicial
Evaluar la población inicial
Repetir
    Aplicar operador de mutación
    Evaluar cada hijo
    Realizar la selección
Mientras no se cumpla la condición de finalización
```

El algoritmo de la programación evolutiva es el mostrado en el Algoritmo 1. La población inicial es de tamaño μ. El operador de mutación obtiene un hijo por cada individuo (es decir, se obtienen μ hijos en total), y está basado en una variable aleatoria con distribución normal. Adviértase que no se aplica recombinación en este caso, porque la programación evolutiva simula la evolución a nivel de las especies, y especies distintas no pueden recombinarse entre sí.

La selección se realiza por medio de una serie de torneos estocásticos, los cuales toman en cuenta los μ padres y los μ hijos. Al final del proceso de selección se tienen nuevamente μ individuos que pasarán a la siguiente generación.

- Estrategias evolutivas.

  Desarrolladas por Ingo Rechenberg [114], quien primero propuso la estrategia evolutiva $(1 + 1)$, que evolucionaba un solo individuo. Posteriormente aparecieron las estrategias evolutivas (μ, λ) y $(\mu + \lambda)$, que ya utilizan una población de más de un individuo.

  El algoritmo de las estrategias evolutivas es el Algoritmo 2. La población inicial es de tamaño μ, y los operadores de recombinación y mutación generan λ hijos.

  Si durante la selección sólo se toman en cuenta los λ hijos para formar la nueva población, se trata de una estrategia evolutiva (μ, λ). Si la selección toma en cuenta tanto padres como hijos, es una estrategia evolutiva $(\mu + \lambda)$.

  El mecanismo de selección es determinístico, a diferencia de la pro-

Algoritmo 2 Estrategias evolutivas

Generar población inicial
Evaluar población inicial
Repetir
    Aplicar operador de mutación
    Aplicar operador de recombinación
    Evaluar cada hijo
    Realizar la selección
Mientras no se cumpla la condición de finalización

gramación evolutiva, por lo que los mejores μ individuos pasan a la siguiente generación. El principal operador es el de mutación, y se basa en una variable aleatoria con distribución normal, al igual que la programación evolutiva. El operador de recombinación es secundario, lo que significa que tiene menos importancia que la mutación y, de hecho, puede omitirse.

- Algoritmos genéticos.

  Propuestos por John Holland [62] con el objetivo de resolver problemas de aprendizaje de máquina.

Algoritmo 3 Algoritmo genético

Generar la población inicial
Evaluar la población inicial
Repetir
    Realizar la selección
    Aplicar operador de recombinación
    Aplicar operador de mutación
    Evaluar cada hijo
Mientras no se cumpla la condición de finalización

El algoritmo genético se muestra en el Algoritmo 3.

En los algoritmos genéticos por lo regular se codifican los individuos en forma de una cadena binaria, a diferencia de la programación evo-

lutiva y las estrategias evolutivas, que no requieren codificación alguna. Hasta ahora han sido propuestas una gran variedad de formas de representación, métodos de selección, y operadores de recombinación y de mutación para los algoritmos genéticos [2].

Nótese, sin embargo, que en el algoritmo genético la recombinación es el operador principal y la mutación es un operador secundario.

1.3 Incorporación de Conocimiento del Dominio en Computación Evolutiva

Los algoritmos evolutivos fueron ideados, y son ampliamente aplicados en problemas con espacios de búsqueda muy grandes. Es común que los espacios de búsqueda sean demasiado grandes para los métodos de optimización tradicionales, debido a que no se tiene mayor conocimiento del problema, o el conocimiento es muy difícil de codificar dentro del método que se trata de aplicar.

Cuando se diseña una heurística de búsqueda, se debe elegir entre flexibilidad y velocidad. Las técnicas que favorecen la velocidad son los sistemas basados en conocimiento, mientras que en el extremo de la flexibilidad se encuentra la búsqueda exhaustiva. La computación evolutiva ha prestado atención a la flexibilidad, resolviendo una gran variedad de problemas difíciles, pero añadiendo una pequeña cantidad de conocimiento del dominio: comúnmente el único conocimiento que se tiene es que se pueden comparar dos soluciones y decir cuál es mejor [112].

Entre los primeros intentos por añadir conocimiento del dominio (además de poder comparar dos soluciones) en computación evolutiva está el algoritmo llamado EnGENEous, de Powell et al. [108], en el que se acopla un sistema experto a un algoritmo genético, aunque resultó muy ineficiente en ciertas aplicaciones. Powell et al. continuaron con la integración de sistemas expertos y algoritmos genéticos en el método llamado "Interdigitación" [110], en el que también incorporaron técnicas de optimización numérica para aproximar mejor el óptimo.

Louis y Rawlins proponen incorporar conocimiento del dominio en varios problemas de diseño, entre ellos el diseño de circuitos lógicos y el diseño de armaduras. Primero la adición del conocimiento fue solamente en los operadores de recombinación [89, 90], y posteriormente el conocimien-

to se incorporó en la iniciación de la población, en la codificación de los individuos y en los operadores evolutivos [91].

Robert Reynolds propuso un tipo de algoritmos evolutivos, en los que el conocimiento del dominio no se integra *a priori* a la técnica, sino que se extrae durante el mismo proceso de búsqueda. Estos son los algoritmos culturales [117].

1.4 Algoritmos Culturales

Los algoritmos culturales fueron desarrollados por Robert G. Reynolds, como un complemento a la metáfora que usan los algoritmos de computación evolutiva, que se habían concentrado en conceptos genéticos, y de selección natural [117].

Los algoritmos culturales están basados en las teorías de algunos sociólogos y arqueólogos, que han tratado de modelar la evolución cultural. Tales investigadores indican que la evolución cultural puede ser vista como un proceso de herencia en dos niveles: el nivel micro-evolutivo, que consiste en el material genético heredado por los padres a sus descendientes, y el nivel macro-evolutivo, que es el conocimiento adquirido por los individuos a través de las generaciones, y que una vez codificado y almacenado, sirve para guiar el comportamiento de los individuos que pertenecen a una población [116, 42].

La cultura puede verse como un conjunto de fenómenos ideológicos compartidos por una población, pero por medio de los cuales, un individuo puede interpretar sus experiencias y decidir su comportamiento. En estos modelos se aprecia muy claramente la parte del sistema que es compartida por la población: el conocimiento, recabado por miembros de la sociedad, pero codificado de tal forma que sea potencialmente accesible a todos. De igual manera se distingue la parte del sistema que es individual: la interpretación de ese conocimiento codificado en forma de un conjunto de símbolos, y los comportamientos que trae como consecuencia su asimilación; también la parte individual incluye las experiencias vividas, y la forma en que éstas pueden aportar algo al conocimiento compartido.

Reynolds intenta captar ese fenómeno de herencia doble en los algoritmos culturales [117]. El objetivo es incrementar las tasas de aprendizaje o convergencia, y de esta manera, que el sistema responda mejor a un gran número de problemas [51].

Los algoritmos culturales operan en dos espacios. Primero, el espacio de la población, como en todos los métodos de computación evolutiva, en el que se tiene un conjunto de individuos. Cada individuo tiene un conjunto de características independientes de los otros, con las que es posible determinar su aptitud. A través del tiempo, tales individuos podrán ser reemplazados por algunos de sus descendientes, obtenidos a partir de un conjunto de operadores aplicados a la población.

El segundo espacio es el de creencias, donde se almacenarán los conocimientos que han adquirido los individuos en generaciones anteriores. La información contenida en este espacio debe ser accesible a cualquier individuo, quien puede utilizarla para modificar su comportamiento.

Para unir ambos espacios se establece un protocolo de comunicación que dicta las reglas del tipo de información que se debe intercambiar entre los espacios.

Dicho todo lo anterior, podemos ver el pseudo-código de un algoritmo cultural en el Algoritmo 4.

Algoritmo 4 Algoritmo cultural

```
Generar población inicial
Iniciar el espacio de creencias
Evaluar población inicial
Repetir
    Actualizar el espacio de creencias (con los individuos aceptados)
    Aplicar operadores de variación (bajo la influencia del espacio de creencias)
    Evaluar cada hijo
    Realizar la selección
Mientras no se cumpla la condición de finalización
```

La mayoría de los pasos de un algoritmo cultural corresponden con los de los algoritmos tradicionales de computación evolutiva, y se puede apreciar que las diferencias están en los pasos que incluyen al espacio de creencias. Por ejemplo, en los primeros pasos se encuentra la iniciación

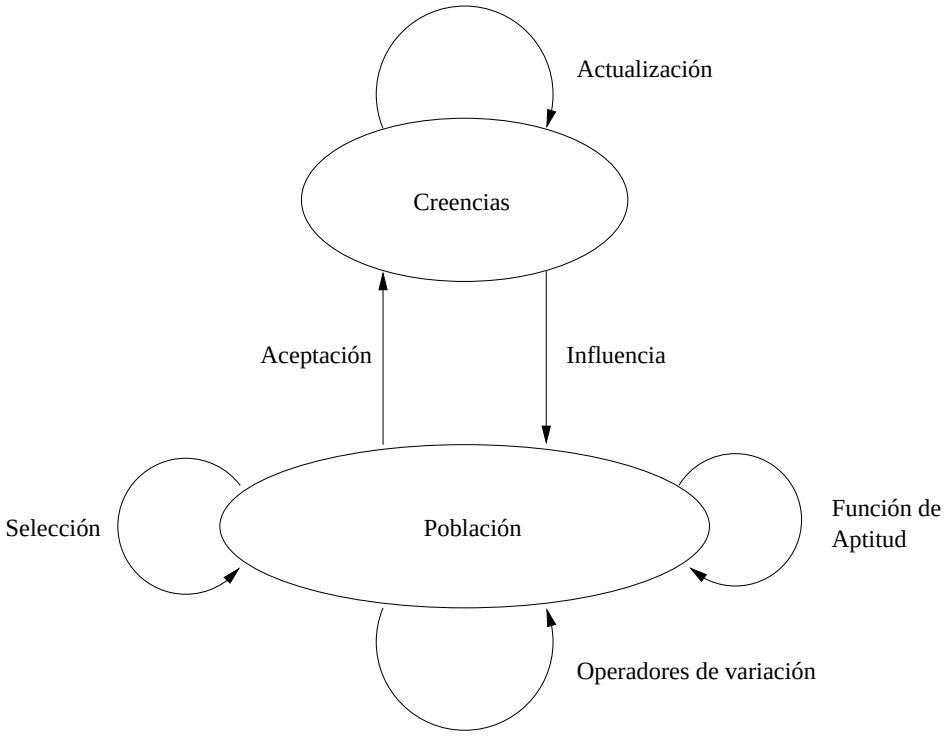

Figura 1.1: Espacios de un algoritmo cultural

del espacio de creencias.

En el ciclo principal, está la actualización del espacio de creencias. Es en ese momento donde el espacio de creencias incorpora las experiencias individuales de un grupo selecto de individuos. Tal grupo se obtiene entre toda la población con la función de *aceptación*.

Por otro lado, los operadores de variación de los individuos (como la recombinación o la mutación) son modificados por la función de *influencia*. La función de influencia ejerce cierta presión, para que los hijos resultantes de la variación se acerquen a los comportamientos deseables, y se alejen de los indeseables, según la información almacenada en el espacio de creencias.

Estas dos funciones, la de aceptación y la de influencia, son mediante las cuales se establece la comunicación entre los espacios de la población y de creencias. Estas interacciones entre los espacios pueden apreciarse en la Figura 1.1 [118].

En [117], Reynolds utiliza la población de un algoritmo genético (junto con sus operadores) como espacio de la población, y los Espacios de Versiones [98] como espacio de creencias. A este algoritmo cultural, Reynolds lo llamó Algoritmo Genético guiado por un Espacio de Versiones (*Version Space guided Genetic Algorithm*, *VGA*).

En la teoría de los algoritmos genéticos existe una expresión, llamada teorema de los esquemas [63], que representa una cota para la velocidad a la que se propagan los mejores esquemas entre la población. Reynolds utilizó algoritmos genéticos, para aprovechar esa expresión y, mediante una breve discusión, intuir cómo un espacio de creencias puede afectar el teorema. De ahí se concluye que, agregando un espacio de creencias a alguno de los métodos de computación evolutiva, es posible mejorar su desempeño, al aumentar las velocidades de convergencia.

En este trabajo se intenta retomar esas afirmaciones de mejora en el desempeño de los algoritmos evolutivos, pero esta vez, en problemas que han sido poco o nunca tratados con algoritmos culturales. Tales problemas son la optimización con restricciones y la optimización multiobjetivo.

Capítulo 2

Trabajo Previo en Optimización Mono-Objetivo con Restricciones

2.1 Funciones de Penalización

La forma más común para tratar problemas de optimización con restricciones en computación evolutiva han sido las funciones de penalización. Dichas funciones consisten en modificar la función objetivo, sumándole o restándole un término que depende de las restricciones violadas. De esta forma, se transforma el problema con restricciones a uno sin restricciones.

Existen dos tipos de métodos de penalización: los interiores y los exteriores. Los métodos interiores trabajan únicamente dentro de la región factible, y dan una penalización pequeña a las soluciones que se encuentran lejos de la frontera de la región factible, mientras que la penalización aumenta conforme nos acercamos a tal frontera. Los métodos exteriores pueden comenzar en zonas infactibles, y la penalización será menor conforme se acerquen a la región factible [111].

Los métodos interiores no son muy comunes, porque tienen la gran desventaja de requerir una solución factible inicial. Es por ello que sólo mencionaremos métodos exteriores de penalización.

La función objetivo modificada, que se utiliza generalmente para los métodos exteriores de penalización es la siguiente:

$$\phi\left(\vec{x}\right) = f\left(\vec{x}\right) \pm \left(\sum_{i=1}^{n} r_i G_i\left(g_i\left(\vec{x}\right)\right) + \sum_{j=1}^{p} c_j H_j\left(h_j\left(\vec{x}\right)\right)\right)$$

donde ϕ es la nueva función objetivo, f es la función objetivo sin modificar, G_i es una función para cada restricción de desigualdad g_i, H_j es una función para cada restricción de igualdad h_j, r_i y c_j son factores de penalización para cada restricción.

Coello [21] menciona las funciones más comunes para G_i y H_j:

$$G_i\left(g_i\left(\vec{x}\right)\right) = \max\left(0, g_i\left(\vec{x}\right)\right)^{\beta}$$
$$H_j\left(h_j\left(\vec{x}\right)\right) = \left|h_j\left(\vec{x}\right)\right|^{\gamma}$$

donde los exponentes β y γ comúnmente son 1 ó 2 ($\beta, \gamma \in \{1, 2\}$).

Richardson et al. [120] estudiaron la forma en que se deben diseñar las funciones de penalización, y trataron de descubrir los aspectos más importantes que se deben tomar en cuenta. Estos son algunos lineamientos reportados en su trabajo:

1. Las funciones de penalización que dependen de la distancia a la zona factible tienen mejor desempeño que aquellas que únicamente dependen del número de restricciones violadas.

2. En problemas que tienen pocas restricciones, y una región factible pequeña, las funciones de penalización que sólo dependen del número de restricciones violadas probablemente no producirán soluciones factibles.

3. Las buenas funciones de penalización deben considerar tanto el costo de cumplimiento máximo, como el costo de cumplimiento esperado (el costo de cumplimiento se refiere a la distancia a la región factible).

4. Las funciones de penalización deben estar cerca del costo de cumplimiento esperado, pero sin ser inferiores a él frecuentemente. En caso contrario, es posible que no se encuentren soluciones factibles.

Se pueden clasificar las funciones de penalización según la información que utilicen para cambiar en el tiempo. El primer tipo de funciones son las estáticas, las cuales se mantienen constantes durante todo el proceso evolutivo. Por lo regular, estas técnicas requieren parámetros (como los factores de penalización) que dependen del problema por resolver.

Por otro lado, si la función de penalización toma como parámetro a la generación actual, se trata de una función dinámica. Estas técnicas normalmente comienzan el proceso evolutivo con penalizaciones bajas, para estimular la búsqueda, y concluyen con penalizaciones altas, de forma que las soluciones finales sean factibles. Kazarlis y Petridis [74] experimentaron con varios tipos de funciones dinámicas, y concluyeron que las cuadráticas consiguen el mejor desempeño.

En esta misma categoría de funciones dinámicas están los métodos de penalización basados en el recocido simulado (por ejemplo, [93]), que incrementan la penalización durante el proceso evolutivo según un horario de enfriamiento, y con una fórmula similar a la del recocido simulado [76].

Las funciones de penalización dinámicas tienen, por lo general, los mismos problemas que las funciones estáticas con los parámetros que dependen del problema.

Otro tipo de funciones de penalización son las funciones adaptativas, en las que la penalización cambia de acuerdo con las condiciones de la población. Por ejemplo [4, 56], donde el factor de penalización se incrementa si el mejor individuo de las últimas generaciones siempre fue factible, se decrementa si tal individuo fue siempre infactible, y se mantiene igual si algunas veces fue factible y otras infactible.

Existen otros tipos de funciones de penalización que son difíciles de clasificar. Por ejemplo, la penalización co-evolutiva de Coello [20], donde se tienen dos poblaciones distintas: la primera, de los individuos que codifican las variables de decisión, y que representan soluciones al problema (ésta es la población normal del algoritmo evolutivo); y la segunda donde se codifican los factores de penalización. De esta forma, se evolucionan al mismo tiempo las soluciones al problema y los factores de la función de penalización. En este algoritmo la función de penalización depende tanto del número de restricciones violadas como de la magnitud de la violación.

Otro algoritmo que utiliza funciones de penalización es el algoritmo genético segregado de Le Riche et al. [121]. Aquí se utilizan dos factores de penalización por cada restricción, uno alto y el otro bajo, en dos sub-poblaciones que interactúan entre sí. Se utilizan dos factores de penalización con el fin de equilibrar las penalizaciones severas con las débiles.

2.2 Otras Técnicas para Manejo de Restricciones en Computación Evolutiva

Además de las funciones de penalización, algunos investigadores han propuesto otras técnicas competitivas.

A continuación se presenta una breve revisión de algunos métodos importantes, y separados según la clasificación de Coello [21].

2.2.1 Representaciones y Operadores Especiales

Algunos problemas pueden ser particularmente difíciles al ser tratados con las representaciones normales de los algoritmos evolutivos. Es esos casos puede ser más fructífero idear una representación especial para el problema y, de ser necesario, un conjunto de operadores especiales aplicables a dicha representación.

Lawrence Davis [30] presenta varios ejemplos de representaciones y operadores especiales para resolver problemas difíciles del mundo real, como generar trayectorias de robots, optimización de horarios, síntesis de la arquitectura de redes neuronales y análisis de ADN.

James Bean utilizó una representación especial, llamada llaves aleatorias [3], con la que resuelve problemas como la asignación de tareas en máquinas paralelas. El objetivo de utilizar las llaves aleatorias es que no sea necesario utilizar operadores especiales, porque mantiene la factibilidad de cualquier solución producida.

Un ejemplo de operadores especiales para espacios convexos es la primera versión de GENOCOP [94], el cual comienza con al menos una solución factible, y los operadores generan hijos que son combinaciones lineales de los individuos originales. Tales operadores son útiles porque el algoritmo está diseñado para trabajar únicamente con problemas con restricciones lineales. Posteriormente se desarrollaron otras versiones de GENOCOP, que además pueden manejar restricciones no lineales, pero tales propuestas no pertenecen a la clasificación de operadores especiales.

Otra propuesta que cuenta con operadores especiales es el algoritmo genético consistente con las restricciones de Kowalczyk [79]. Utiliza operadores especiales, así como un método especial de iniciación de la población para asegurar que los individuos en la población serán consistentes

con las restricciones. Sin embargo, tales operadores y el método de iniciación son muy costosos computacionalmente.

Schoenauer y Michalewicz [126] han utilizado operadores que tratan de explorar la frontera entre las zonas factible y no factible. La motivación para desarrollar este tipo de operadores, es que en muchos problemas de optimización no lineal el óptimo global está ubicado en la frontera de la zona factible. El problema es que estos operadores son altamente especializados, al grado de tener que ser diseñados específicamente para un problema en particular.

Un tipo de técnicas importantes que se basan en representaciones especiales son los decodificadores, los cuales son una transformación que relaciona una solución factible con una solución codificada. La transformación debe ser construida de tal forma que cualquier posible solución producida por el algoritmo pueda ser transformada en una solución factible. La técnica más importante hasta la fecha que utiliza decodificadores, es la de los mapas homomorfos de Koziel y Michalewicz [80, 81], que relaciona un cubo n-dimensional con la región factible de algún problema. Una ventaja sobresaliente de esta técnica es que puede manejar cualquier tipo de restricciones, con zonas factibles convexas y no convexas, e incluso disjuntas. Esta propuesta es importante, por los resultados publicados de un grupo de funciones de prueba (propuesto por Michalewicz y Schoenauer, [96]), que fueron los mejores producidos por cualquier algoritmo evolutivo hasta ese momento, y que por tal motivo se han convertido en un punto de comparación para los algoritmos evolutivos con mecanismos para manejo de restricciones.

2.2.2 Algoritmos de Reparación

Los algoritmos de reparación son muy utilizados en problemas optimización combinatoria, en donde se convierten permutaciones inválidas en permutaciones válidas. Además, se ha mostrado que, en problemas de optimización combinatoria, estos algoritmos presentan un muy buen desempeño [85, 86].

En una de las versiones más recientes de GENOCOP (el GENOCOP III [95]), también se utilizó la reparación de las soluciones. En esta versión se utilizan dos sub-poblaciones. La primera de ellas contiene los puntos de búsqueda, y es similar a la población original de GENOCOP, donde

se mantiene la factibilidad con respecto a las restricciones lineales. La segunda sub-población contiene algunos puntos factibles de referencia. Los puntos en la primera población son reparados antes de hacer la evaluación.

2.2.3 Separación de Restricciones y Objetivos

Algunos enfoques tratan de forma distinta a los objetivos y a las restricciones. Por ejemplo, Paredis [103] propuso un algoritmo co-evolutivo, pero no para penalización. Utiliza dos sub-poblaciones, una representa las soluciones del problema, y la otra representa a las restricciones. Cada individuo de la segunda sub-población es evaluado según la cantidad de individuos en la primera población que violan la restricción que representa.

Varios autores han propuesto métodos en los que se trata como superiores a los puntos factibles. Por ejemplo, Powell y Skolnick [109] propusieron un método donde los puntos factibles siempre tienen una aptitud entre $-\infty$ y 1, y los puntos infactibles siempre tienen una aptitud entre 1 y $+\infty$.

Deb también tiene un método similar [35], en el que los puntos infactibles siempre tienen una peor aptitud que los puntos factibles, porque se agrega la magnitud de las violaciones a la aptitud del peor individuo factible. Para hacer más simple esta asignación de aptitud, Deb utiliza torneos binarios, en los que las reglas son las siguientes:

1. Si un individuo es factible y el otro infactible, siempre gana el individuo factible.

2. Si ambos individuos son factibles, gana el que tenga mejor valor de la función de aptitud.

3. Si ambos individuos son infactibles, gana el individuo que tenga una magnitud más baja en la violación de restricciones.

Hinterding y Michalewicz [61] propusieron un método que complementa a los padres antes de hacer la recombinación. La selección de ambos padres se hace con criterios distintos: el primer padre se selecciona tomando en cuenta su aptitud y su factibilidad, mientras que el segundo se selecciona de forma que tenga el mínimo de restricciones satisfechas en común

con el primero. Esta idea intenta producir descendientes que sean mejores que los padres, en el sentido de restricciones violadas.

Hay más métodos que tratan en forma diferente a los objetivos y a las restricciones. Schoenauer y Xanthakis [127] extendieron un método llamado memoria conductista, para hacerlo capaz de manejar restricciones. Este método trabaja en varias fases, y en cada una se trata de optimizar una de las restricciones, hasta que en la fase final se trata de optimizar la función objetivo, suponiendo que se tienen suficientes soluciones factibles en la población. Este método puede ser bastante sensible al orden en el que se optimizan las restricciones.

También hay varios enfoques que utilizan técnicas de optimización multiobjetivo. Dado que cada una de las restricciones puede tratarse como un objetivo en el que hay que minimizar la violación, podemos agregar el objetivo original al conjunto de restricciones, siendo todas estas funciones los objetivos por optimizar.

COMOGA [131, 130], de Surry et al., utiliza el algoritmo evolutivo multiobjetivo llamado VEGA [125], así como jerarquización de Pareto. Parmee y Purchase [105] también utilizan VEGA en un problema del mundo real. Camponogara y Talukdar [9] convierten el problema de optimización con restricciones en un problema biobjetivo, donde un objetivo es el objetivo original, y el otro es la magnitud de la suma de las violaciones a las restricciones. Jiménez y Verdegay [70] utilizan el método min-max, con unas reglas para la selección muy parecidas al método de Deb [35].

Coello ha propuesto varios métodos basados en técnicas multiobjetivo. El primero [19] está basado en sub-poblaciones, como VEGA, y en cada sub-población se trata de minimizar la violación a una de las restricciones.

Posteriormente, Coello propuso otro algoritmo [18] pero esta vez basado en no dominancia, y que jerarquiza a los individuos según la cantidad de puntos infactibles en la población.

Más recientemente, Coello y Mezura tienen otro algoritmo [17], en el que la selección se hace mediante torneos basados en la no dominancia de las soluciones.

Ray et al. [113] propusieron un método donde se jerarquiza a los individuos con base en su valor de las funciones objetivo y a la violación de restricciones. Posteriormente se utiliza un mecanismo de complemento de restricciones como el de Hinterding y Michalewicz [61].

Finalmente, Runarson y Yao [123] propusieron un método llamado jerarquización estocástica, en el que la jerarquización se realiza por medio de

una versión estocástica del ordenamiento de la burbuja. Se utiliza una estrategia evolutiva para el proceso de optimización. Este método es importante, porque para validarlo se utilizó el conjunto de funciones de prueba de Michalewicz y Schoenauer [96], y en los resultados igualó o mejoró a los mapas homomorfos [81], con un costo computacional menor.

2.2.4 Métodos Híbridos

Los métodos híbridos son aquellos en los que se combinan dos o más algoritmos distintos de optimización.

El primer método híbrido que mencionaremos, es el propuesto por Adeli y Cheng [1], que utiliza el método de minimización de Lagrange combinándolo con una función de penalización.

Kim y Myung [75, 99] propusieron un método de dos fases, en el cual la primera fase es muy similar al método de Adeli y Cheng.

Otro método híbrido es CORE [5] (*Constrained Optimization by Random Evolution*, Optimización con Restricciones mediante Evolución Aleatoria), que utiliza algún método de programación matemática para optimización sin restricciones, combinado con una búsqueda evolutiva aleatoria para minimizar la violación de restricciones.

También se ha usado lógica difusa como ingrediente en un método basado en computación evolutiva. En el trabajo de Van Le [84] se reemplazan las restricciones originales por restricciones difusas, definidas por él mismo. Como algoritmo evolutivo de optimización se utiliza la programación evolutiva [47].

La simulación del sistema inmune es una heurística reciente que se ha aplicado a optimización con restricciones. El sistema tiene una población de antígenos y otra de anticuerpos. Los antígenos son agentes indeseables, mientras que los anticuerpos son agentes que tratan de parecerse lo más posible a los antígenos, para poder neutralizarlos. Las primeras propuestas para manejar restricciones con una simulación del sistema inmune fueron hechas por Hajela y Lee [57, 58], donde designaban antígenos a algunos individuos factibles. Las estrategias de expresión son una versión más simple de las técnicas de simulación del sistema inmune propuestas anteriormente [60]. Más recientemente, Coello y Cruz [16] propusieron una paralelización de la simulación del sistema inmune para manejar problemas con restricciones.

Dorigo et. al. [41, 40, 39, 38] propusieron la técnica de optimización usando una colonia de hormigas, basándose en el comportamiento de las colonias de hormigas verdaderas. Las propuestas para manejar restricciones con la técnica de optimización usando la colonia de hormigas fueron hechas por Bilchev y Parmee [6, 7], que definieron a algunas fuentes de comida como inaceptables, cuando son violadas algunas restricciones.

2.3 Algoritmos Culturales para Manejo de Restricciones

Los algoritmos culturales son considerados métodos híbridos, pero se encuentran separados en esta sección por la relevancia que tienen en este trabajo.

Entre las primeras propuestas para realizar optimización con restricciones con algoritmos culturales, está la utilización del algoritmo llamado GENOCOP [94], al que se le agregó un espacio de creencias para convertirlo en un algoritmo cultural, trabajo realizado por Reynolds et al. [119]. Posteriormente, Chung y Reynolds también utilizaron la población de la programación evolutiva como espacio de la población [13], empleando el mismo tipo de espacio de creencias que se usó con GENOCOP. Esta generación de algoritmos culturales fue desarrollada para utilizarse con problemas de evaluación en espacios continuos [118],

El espacio de creencias utilizado con estos algoritmos está basado en los esquemas de intervalos de Eshelman y Schaffer [44], tratando de representar de forma numérica intervalos para cada una de las variables de decisión, definidos por las restricciones del problema. En este caso, los intervalos reflejan la información adquirida por la población en cuanto a la región factible, que en un principio son los intervalos dados en los datos de entrada, y se actualizan con los nuevos individuos que aparecen en la población durante el proceso evolutivo. Adviértase que de esta manera, en el espacio de creencias se está representando un espacio de búsqueda con una región factible convexa.

No obstante que el espacio de creencias es similar, la función de influencia debe ser distinta para cada espacio de población, debido a que cada uno utiliza operadores evolutivos distintos.

Como vimos al describir a GENOCOP, su operador de recombinación

es una combinación lineal de los padres, al cual se le exige ahora que el mejor padre esté dentro del espacio de creencias (es decir, dentro de los intervalos almacenados). También se modifican varios tipos de mutación utilizados en GENOCOP, pero la idea central en todos ellos es tratar de mover a los individuos mutados dentro del espacio de creencias.

En la programación evolutiva, dado que se tiene únicamente el operador de mutación, éste se modificó para que el individuo mutado se acerque a la región determinada por el espacio de creencias y lo abarque lo mejor posible. Además se tiene un operador de reparación, en el caso en el que el individuo generado sea infactible, para introducirlo de nuevo entre los intervalos almacenados en el espacio de creencias.

2.3.1 CAEP

Basados en su trabajo anterior, Chung y Reynolds desarrollaron los algoritmos llamados CAEP (*Cultural Algorithms with Evolutionary Programming*, Algoritmos Culturales con Programación Evolutiva) [14], probablemente motivados por el bajo costo computacional mostrado por la programación evolutiva al compararla con GENOCOP, en las versiones culturales de ambos algoritmos [13].

La estructura básica de un CAEP se muestra en el Algoritmo 5 [12].

Los intervalos donde se encuentran buenas soluciones fueron conservados dentro del espacio de creencias. A estos esquemas de intervalos se les llamó conocimiento normativo, porque dictan las normas dentro de las cuales se espera que estén la mayoría de los individuos.

Pero también se incorporó el conocimiento situacional al espacio de creencias, que almacena los puntos exactos de los mejores individuos encontrados hasta el momento. De esta forma, los mejores individuos se convierten en líderes a seguir.

Chung propone varias formas en las que el conocimiento situacional, el conocimiento normativo, o ambos, pueden influir en el operador de mutación [12]. Sin embargo, el trabajo de Chung con los CAEP se concentra en optimización sin restricciones. Jin y Reynolds son quienes comienzan a utilizar los CAEP para optimización con restricciones [71].

Jin y Reynolds utilizan la parte normativa del espacio de creencias de Chung, pero eliminan la parte situacional y la sustituyen por un conocimiento de restricciones, que es un esquema basado en regiones. El cono-

Algoritmo 5 Estructura básica de un CAEP

Generar la población inicial de tamaño p

Evaluar la población inicial

Iniciar el espacio de creencias

Repetir

Aplicar el operador de mutación bajo la influencia del espacio de creencias para generar p hijos (ahora hay $2p$ individuos en la población)

Evaluar cada hijo

Realizar los torneos binarios, eligiendo aleatoriamente c contrincantes para cada individuo

Seleccionar los p individuos con mayor número de victorias en los torneos, para formar la población de la siguiente generación

Actualizar el espacio de creencias con los individuos aceptados

Mientras no se cumpla la condición de finalización

cimiento de restricciones en el espacio de creencias, toma la región determinada por los intervalos almacenados en el conocimiento normativo y divide esa región en celdas, llamadas celdas de creencias.

Las celdas, basadas en las características de los individuos en la población, indican si la región es factible o no factible, mediante el tipo de la celda.

La parte de restricciones del espacio de creencias se encarga de crear una especie de mapa de navegación, para guiar a los individuos a la zona factible. Como las celdas son independientes entre sí, la zona factible puede ser no convexa, e incluso puede ser disjunta, sin que esto sea una dificultad para el manejo del conocimiento de restricciones.

Pero Jin y Reynolds no dan ninguna indicación para la implementación de este espacio de creencias, y un conjunto estático de celdas puede ser muy ineficiente en problemas de alta dimensionalidad. Además del conocimiento de restricciones en el espacio de creencias, no hay mayores cambios en el algoritmo de Chung.

Posteriormente, Jin y Reynolds utilizan este mismo algoritmo para aplicaciones de minería de datos [72]. En esta nueva versión, se menciona el uso de algunas estructuras de datos para representar las celdas de creencias, pero no se dan mayores indicaciones para su implementación. Además, esta propuesta se aleja un poco del tema de optimización con restricciones para adentrarse en la minería de datos.

Las limitantes antes descritas de los algoritmos culturales para optimización con restricciones fueron la motivación principal para realizar este trabajo de tesis.

Capítulo 3

Técnica Propuesta para Optimización Mono-Objetivo con Restricciones

La técnica aquí propuesta es un algoritmo cultural con programación evolutiva, CAEP, cuya estructura básica se muestra nuevamente en el Algoritmo 6.

La población consiste de un conjunto de individuos, y cada uno es una posible solución al problema. Cada individuo contiene las n variables de decisión del problema, las cuales se hallan sin codificar.

Al principio, se genera una población inicial aleatoria de tamaño p. Cada variable se genera mediante una distribución uniforme dentro de los intervalos dados como datos de entrada del problema.

Al evaluar un individuo se obtiene su aptitud mediante la función objetivo, y también se verifica si el individuo es factible o no.

Este algoritmo es una modificación de la técnica de Jin y Reynolds [71], que también utiliza programación evolutiva. En su técnica, Jin y Reynolds utilizaban una estructura estática para representar el espacio de creencias, la cual puede crecer excesivamente en el espacio de memoria requerido, como se verá claramente en la descripción del espacio de creencias que se presenta más adelante.

A continuación se describe el espacio de creencias, así como los pasos restantes del algoritmo.

Algoritmo 6 Estructura básica de un CAEP

Generar la población inicial de tamaño p

Evaluar la población inicial

Iniciar el espacio de creencias

Repetir

Aplicar el operador de mutación bajo la influencia del espacio de creencias para generar p hijos (ahora hay $2p$ hijos en la población)

Evaluar cada hijo

Realizar los torneos binarios, eligiendo aleatoriamente c contrincantes para cada individuo

Seleccionar los p individuos con mayor número de victorias en los torneos, para formar la población de la siguiente generación

Actualizar el espacio de creencias con los individuos aceptados

Mientras no se cumpla la condición de finalización

l_1	u_1	l_2	u_2	$\cdots$	l_n	u_n
L_1	U_1	L_2	U_2	$\cdots$	L_n	U_n

Figura 3.1: Parte normativa del espacio de creencias

3.1 Estructura del Espacio de Creencias

El espacio de creencias tiene dos tipos de conocimiento: el normativo y el de restricciones.

La parte normativa del espacio de restricciones contiene las cotas inferiores, l_i, y las cotas superiores, u_i, de los intervalos prometedores para cada una de las n variables de decisión del problema, x_i con $i = 1, \ldots, n$. Además, contiene la aptitud del mejor individuo hallado en cada una de esas cotas, llamados L_i para la cota l_i, y U_i para la cota u_i. Estos componentes se muestran en la Figura 3.1.

La parte de restricciones del espacio de creencias, está formada por un conjunto de celdas de creencias, que son divisiones del espacio de búsqueda en los intervalos almacenados en la parte normativa. El objetivo de dividir el espacio de búsqueda es hacer una especie de mapa de navegación, donde se conozca el tipo de región en cuanto a la factibilidad.

Cada celda tiene un tipo, que depende de la factibilidad de la región que ocupa. En la Figura 3.2 se muestra la región factible de un problema, así como su representación en la parte de restricciones, incluyendo los tipos de celda. Cada celda c tiene los siguientes campos:

- El tipo de la celda, $tipo_c$.
- El número de individuos factibles que han caído dentro de esa celda, nf_c.
- El número de individuos infactibles que han caído dentro de esa celda, ni_c.
- Las cotas inferiores y superiores de los intervalos para cada una de las variables de decisión de la región que representa la celda, $l_{i,c}$ y $u_{i,c}$, respectivamente, con $i = 1, \ldots, n$.

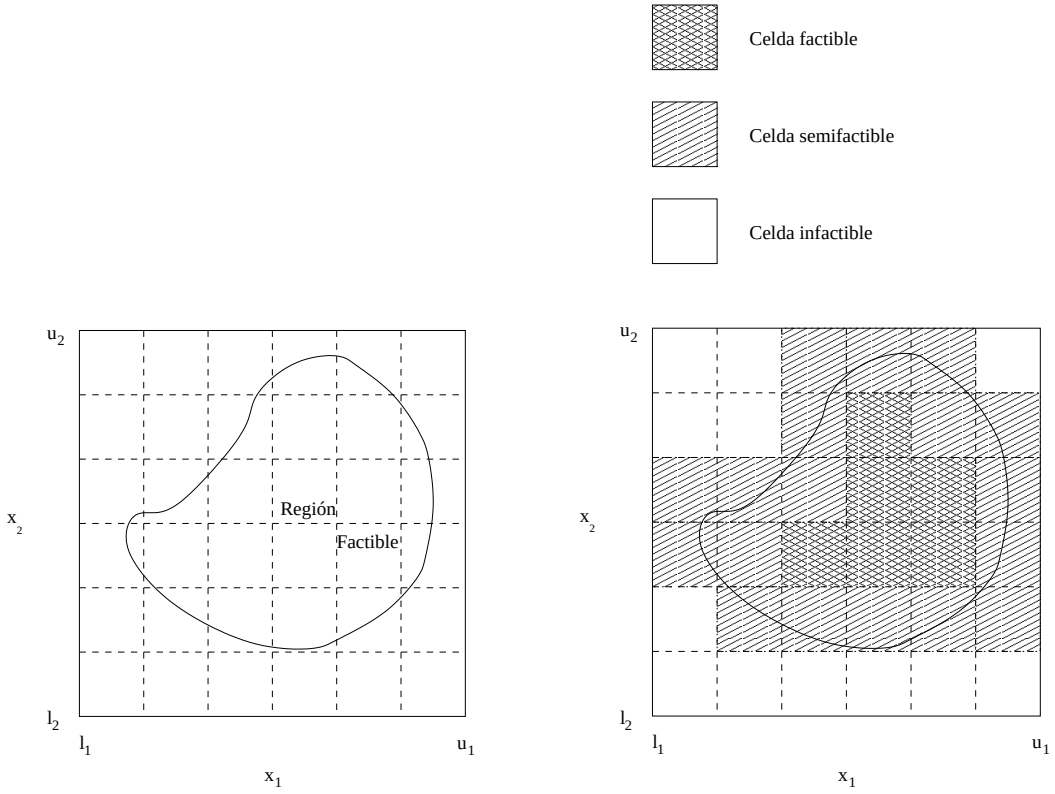

Figura 3.2: Parte de restricciones del espacio de creencias

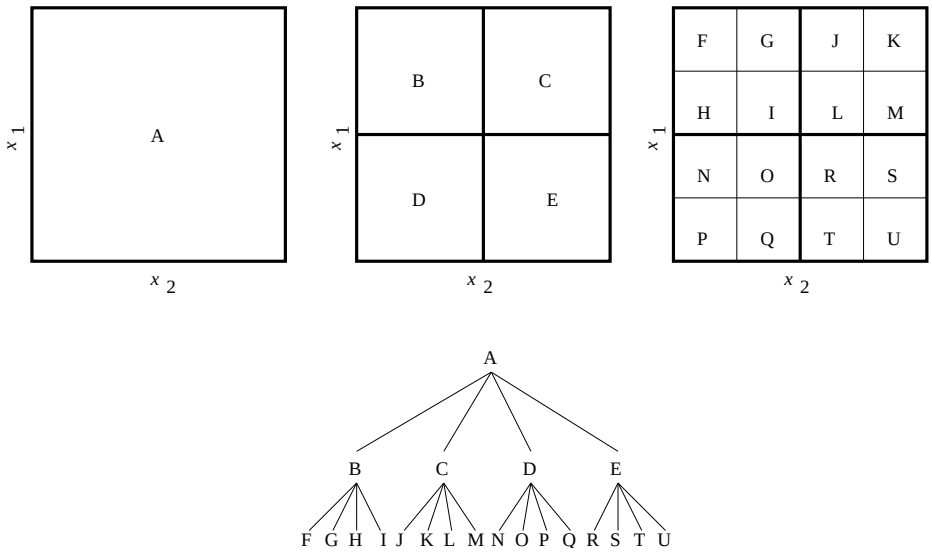

Figura 3.3: *Quadtree*

3.2 Celdas de Creencias y 2^m-*trees*

Los 2^m-*trees* son estructuras de datos espaciales, muy utilizadas, por ejemplo, en planeación de movimientos de robot [83] y en gráficos por computadora. Son árboles donde cada nodo que no es hoja, tiene exactamente 2^m hijos.

Cada nodo en un 2^m-*tree* representa una región de un espacio m-dimensional, y si un nodo se divide y se le asignan hijos, dicha división se realiza siempre por la mitad en cada dimensión, y cada una de las divisiones resultantes es representada por uno de los hijos. En la Figura 3.3 se muestra un *quadtree* (2^2-*tree*) que, mediante dos divisiones sucesivas, representa un espacio de dos dimensiones.

Los 2^m-*trees* son útiles, porque sólo se dividen los nodos en los que se desea tener mayor detalle en las características de la región que representan, impidiendo con ello que el árbol crezca desmesuradamente, y consuma demasiada memoria en su implementación. En la Figura 3.4 se muestra cómo un *quadtree* representa con mayor refinamiento sólo las áreas que nos interesan en nuestro problema, es decir, la frontera entre las regiones factible y no factible.

En la técnica que se propone en esta tesis se usan 2^m-*trees* para partir el

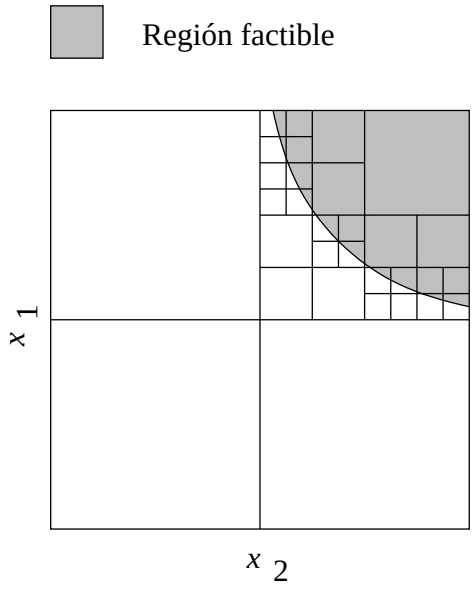

Figura 3.4: División de un espacio bidimensional con una porción de la región factible por un *quadtree*

espacio de búsqueda en celdas, de tal forma que cuando el problema tiene una alta dimensionalidad, no crezca exponencialmente el uso de memoria necesario para la representación del espacio como ocurre con la implementación de Jin y Reynolds [71]. Cada nodo representará una celda.

No obstante, sigue existiendo el problema de la dimensionalidad. No se pueden hacer divisiones en todas las dimensiones del problema si éste tiene muchas variables de decisión, puesto que los 2^m*-trees* tienen utilidad práctica sólo cuando $m \leq 4$ [83].

En la técnica presentada aquí, utilizamos un *octree* (2^3*-tree*) cuando el problema por resolver tiene 3 o más variables de decisión. Cuando el problema tiene dos variables, utilizamos un *quadtree*.

La manera en que utilizamos un 2^m*-tree* con una m menor que el número de variables de decisión del problema, es haciendo una proyección del espacio de búsqueda en m dimensiones cada vez que se tiene que dividir un nodo. Lo ideal sería escoger la proyección que divida mejor el espacio de búsqueda en una región factible y otra no factible, dentro de la región representada por el nodo que se desea dividir, pero el número de combinaciones de las dimensiones que se deben probar para encontrar tal

proyección crece demasiado con la dimensionalidad del problema, lo que hace imposible experimentar con todas ellas.

Por lo tanto, se puede elegir un grupo de proyecciones que se puedan probar, de manera que el número de combinaciones crezca linealmente con la dimensionalidad del problema (para así poder investigar más combinaciones cuando se tiene un problema con más variables).

Para saber elegir cuándo se ha encontrado una proyección que divide bien el espacio de búsqueda en la región de un nodo dado, se debe contar el número de individuos factibles e infactibles en cada uno de sus hijos, y la proyección será mejor mientras alguno de los dos valores tienda a cero, ya que en este caso, el nodo estará casi totalmente dentro de una zona, ya sea la factible o la no factible.

Dicho de otra forma, un número $\min(nf_c, ni_c)$ pequeño indicará que el nodo c está casi totalmente dentro de una de las zonas (factible o no factible), y entonces se tiene una buena proyección. Si deseamos encontrar la mejor proyección para la división de un nodo c, debemos encontrar el conjunto D de las dimensiones que minimizan:

$$\lambda(c, D) = \sum_{h \in Hijos(c,D)} \min(nf_h, ni_h)$$

donde $\lambda(c, D)$ es el número por minimizar para el nodo c, $Hijos(c, D)$ es el conjunto de nodos hijos resultantes de la división de c sólo en las dimensiones dadas por el conjunto D, y D puede ser cualquier conjunto de dimensiones, mientras cumpla $|D| = m$ (para un 2^m-*tree*).

Teniendo la mejor proyección para el nodo c (dentro de un grupo de proyecciones de prueba), podemos continuar dividiendo a sus hijos mediante el mismo método, hasta llegar a la profundidad máxima permitida del 2^m-*tree*. Como se han probado varias proyecciones para hacer una sola división de nodo, sería una mejor elección poner un límite de profundidad pequeño, para no invertir demasiado tiempo en la creación del árbol.

3.3 Iniciación del Espacio de Creencias

3.3.1 Iniciación de la Parte Normativa

La parte normativa del espacio de creencias se inicia poniendo en las cotas de los intervalos de las variables, los valores dados en los datos de entrada

del problema, mientras que las aptitudes para todos los casos serán $+\infty$ (suponiendo que se trata de un problema de minimización):

$$\begin{aligned} L_i &= +\infty \\ U_i &= +\infty \end{aligned}$$

para $i = 1, \ldots, n$.

3.3.2 Iniciación de la Parte de Restricciones

En cuanto a la parte de restricciones, el nodo raíz del 2^m-*tree* contiene los intervalos de la parte normativa, y la primera división de nodo se efectúa hasta que se tienen listos los individuos de la generación cero.

Se eligen m dimensiones al azar, a las cuales representaremos por el conjunto D. Luego se prueba dividiendo el nodo raíz en esas dimensiones de D, y se evalúa el número λ descrito antes, siendo c el nodo raíz. El proceso se repite t veces, almacenando los λ resultantes de cada conjunto de dimensiones D, y al final se elige el conjunto de dimensiones que tenga el valor de λ más pequeño.

En caso de que el problema tenga tres variables, se utilizará un *octree* y no hay proyecciones, por lo que se divide sin evaluar el número λ. Lo mismo ocurre si hay dos variables, puesto que se utilizará un *quadtree* en lugar de un *octree*.

Una vez generado el árbol, a cada celda c se le asigna un tipo, que servirá para guiar el proceso evolutivo. Se tienen cuatro clases posibles:

- si $nf_c = 0$ y $ni_c = 0$, entonces $tipo_c = desconocido$
- si $nf_c > 0$ y $ni_c = 0$, entonces $tipo_c = factible$
- si $nf_c = 0$ y $ni_c > 0$, entonces $tipo_c = infactible$
- si $nf_c > 0$ e $ni_c > 0$, entonces $tipo_c = semifactible$

Finalmente, todas las hojas cuyo tipo de celda sea semifactible, se expanden siguiendo el procedimiento descrito, a menos que se encuentren en la profundidad máxima permitida del árbol. Se expandirán tantos nodos como celdas semifactibles haya, deteniéndose cuando se llegue a la profundidad máxima. Este valor de profundidad máxima depende de la

memoria disponible en el sistema donde se ejecute. Para los experimentos realizados aquí, se utilizó una profundidad máxima de 5, con lo que no puede haber más de $4096 + 512 + 64 + 8 + 1 = 4681$ nodos; este peor caso casi nunca se presenta, y en la práctica se tiene normalmente un número mucho menor de nodos.

3.4 Actualización del Espacio de Creencias

La parte de restricciones del espacio de creencias se actualiza cada generación, mientras que la parte normativa se actualiza a cada $g_{normativa}$ generaciones, donde $g_{normativa}$ es un parámetro dado por el usuario.

3.4.1 Actualización de la Parte de Restricciones

La actualización de la parte de restricciones consiste únicamente en agregar los nuevos individuos que caigan en cada celda c a las variables nf_c y ni_c.

Si resulta que una hoja en una profundidad menor que el límite se convierte en semifactible por esta adición, tal nodo se dividirá con el método descrito anteriormente.

3.4.2 Actualización de la Parte Normativa

La actualización de la parte normativa es más compleja, y es por eso que no se ejecuta en cada generación. Cuando se actualiza el intervalo de cada variable, las celdas de la parte de restricciones cambian, y es necesario reconstruir el árbol desde la raíz. Además, esta actualización se realiza sólo considerando a una parte de la población; tal fracción la selecciona la función de aceptación, tomando como parámetro dado por el usuario el porcentaje de la población total que se usará.

La forma en que se reconstruye el árbol es la misma en la que se construyó por primera vez. Sin embargo, ahora el árbol cubrirá una parte menor del espacio de búsqueda, que es precisamente donde se han encontrado los mejores individuos de las generaciones pasadas.

La actualización de los intervalos se hace para cada una de las variables de decisión del problema. Puede ocurrir de dos formas:

- Un intervalo será ampliado (una generalización) si los individuos aceptados caen fuera de él, en la dimensión que representa.
- Un intervalo será reducido (una especialización) si los individuos aceptados caen dentro y a la vez algún individuo aceptado factible tiene mejor aptitud que las aptitudes almacenadas para las cotas de los intervalos.

Para que las reglas de actualización de la parte normativa funcionen adecuadamente, es necesario que la función de aceptación prefiera individuos factibles, y con aptitudes elevadas como segundo criterio. De no ser así, casi nunca se cumplirá la condición de reducción del intervalo, porque casi nunca serán factibles los individuos aceptados.

3.5 Influencia del Espacio de Creencias en la Mutación

Se realiza mutación gaussiana, como en la programación evolutiva tradicional, pero en este caso existe una autoadaptación de los parámetros, que depende de la información del espacio de creencias.

El operador de mutación funciona obedeciendo las siguientes reglas:

Sea $N(\mu, \sigma)$ una variable aleatoria con una distribución normal con media μ y desviación estándar σ. La mutación se realiza para cada variable x_i de cada individuo x.

- Si $x_i < l_i$ entonces $x'_i = x_i + |N(0, u_i - l_i)|$.
- Si $x_i > u_i$ entonces $x'_i = x_i - |N(0, u_i - l_i)|$.
- Si x está en una celda c, cuyo tipo es semifactible, factible o desconocida, entonces $x'_i = x_i + N(0, u_{i,c} - l_{i,c})$.
- Finalmente, si x está en una celda c, cuyo tipo es infactible, entonces $x'_i = N\left(\frac{l_{i,cercana}+u_{i,cercana}}{2}, u_{i,cercana} - l_{i,cercana}\right)$.

  Donde $cercana$ es la celda semifactible más cercana a c. Sin embargo, si no se encuentra ninguna celda semifactible, $cercana$ será la celda factible más cercana. De nuevo, si no se encuentra una celda factible,

$cercana$ será la celda desconocida más cercana. Si tampoco se encuentra una celda desconocida (lo que implicaría que se tienen únicamente celdas infactibles), entonces se aplica la siguiente mutación: $x_i' = N\left(\frac{l_i+u_i}{2}, u_i - l_i\right)$.

La búsqueda en el árbol para el último caso en la mutación, se hace primero en los hermanos del nodo que se toma como punto de partida. Si en los hermanos no se encuentra una celda con el tipo deseado, se mueve el nodo actual al padre, y se busca entre los hermanos de éste. Si el proceso de búsqueda llega a la raíz, y no se tiene éxito, toda la búsqueda falla, y se procede al siguiente caso.

3.6 Torneo de selección

Las reglas para la actualización del espacio de creencias pueden hacer que los conocimientos se especialicen lentamente, puesto que son estrictas en la reducción de los intervalos.

Para ayudar a la velocidad del algoritmo se pueden aprovechar las reglas del torneo para la selección.

Después de realizar la mutación, se tendrá una población de tamaño $2p$, puesto que p padres generan p hijos, y el torneo se realiza considerándolos a todos.

Los torneos son de c enfrentamientos (binarios) por individuo, con los c contrincantes elegidos al azar de entre toda la población. Al finalizar los torneos, los individuos seleccionados para pasar a la siguiente generación, serán los p individuos con más victorias.

Las reglas para el torneo adoptadas en este trabajo resultaron ser muy parecidas a las de Deb para su penalización con base en la factibilidad [35], pero en nuestro caso nunca se hace suma de restricciones violadas.

Las reglas para el torneo son:

1. Si ambos individuos son factibles, o ambos son no factibles, gana quien tenga un mejor valor en la función objetivo.

2. En otro caso, gana siempre el individuo factible.

De esta forma, se tiene lista la población para realizar todo el proceso en la siguiente generación.

3.7 Parámetros de la Técnica

En la Tabla 3.1 se muestran los parámetros que utiliza esta técnica. Los valores recomendados son de acuerdo con nuestras observaciones, después de numerosos experimentos.

Tabla 3.1: Parámetros de la técnica para optimización con restricciones.

Parámetro	**Significado**	**Valor recomendado**
p	Tamaño de la población.	La técnica funciona mejor con tamaños pequeños de población. Valores que dan buenos resultados están entre 10 y 50.
G_{max}	Número máximo de generaciones.	Dependiendo del problema, se puede obtener una buena aproximación al óptimo con 1000 generaciones, pero puede incrementarse si se desea mayor aproximación.
$g_{normativa}$	Frecuencia, en generaciones, con la que se actualiza la parte normativa.	Un número pequeño aumenta el tiempo de máquina necesario, y un número grande provoca que el conocimiento se especialice lentamente. Un valor medio es 20.
$aceptación$	Porcentaje de individuos aceptados que contribuirán a actualizar el espacio de creencias	Un número grande provoca que el conocimiento se especialice con lentitud. Se recomiendan valores menores a 50%, y de preferencia alrededor de 10%.

Tabla 3.1: Parámetros de la técnica para optimización con restricciones.

Parámetro	**Significado**	**Valor recomendado**
t	Número de pruebas sobre dimensiones distintas para elegir sobre cuáles dividir un nodo del 2^m-*tree*.	Cuando $n = 1$ ó 2, este parámetro no se toma en cuenta (se asigna $t = 1$). Para problemas de dimensionalidad mayor, $t = n$ es un valor adecuado, para no incrementar demasiado el tiempo de máquina necesario para la creación del árbol.
$profundidad_{máx}$	Es la profundidad máxima del árbol.	Al incrementar este valor, aumenta la cantidad de memoria necesaria para almacenar el árbol. 5 es un valor austero, 6 ó 7 aún son valores apropiados, pero más de 7 podría significar demasiada memoria utilizada.
c	Número de encuentros por individuo durante el torneo.	Debe estar entre 1 y $2p - 1$. Un valor pequeño hace más aleatoria a la selección, y un valor grande requiere mayor tiempo de máquina. Con $\frac{p}{2}$ se obtienen buenos resultados.

Capítulo 4

Comparación de Resultados en Optimización Mono-Objetivo con Restricciones

4.1 Funciones de Prueba

Para validar la técnica de manejo de restricciones que se propone en esta tesis, se utilizó un conjunto de 13 funciones de prueba que es estándar en la literatura especializada. Las primeras 11 funciones de prueba fueron propuestas por Michalewicz y Schoenauer en [96], y constituyen el conjunto de prueba más utilizado en optimización evolutiva con restricciones. La función de prueba número 12 fue agregada al conjunto por Koziel y Michalewicz en [81]. La última función de prueba fue tomada de [92].

En este conjunto hay funciones desde dos hasta veinte variables de decisión, con funciones objetivo lineales y no lineales, con una o hasta nueve restricciones, entre ellas hay de igualdad y desigualdad, y las regiones factibles tienen diversas formas y proporciones.

Todas estas funciones de prueba se muestran a continuación.

g01.

Minimizar:

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

sujeto a:

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

donde: $0 \leq x_i \leq 1$ ($i = 1, \ldots, 9$), $0 \leq x_i \leq 100$ ($i = 10, 11, 12$) y $0 \leq x_{13} \leq 1$.

g02.

Maximizar:

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

sujeto a:

$$\begin{aligned} g_1(\vec{x}) &= 0.75 - \prod_{i=1}^{n} x_i \leq 0 \\ g_2(\vec{x}) &= \sum_{i=1}^{n} x_i - 7.5n \leq 0 \end{aligned}$$

donde: $n = 20$ y $0 \leq x_i \leq 10$ ($i = 1, \ldots, n$).

g03.

Maximizar:

$$f(\vec{x}) = (\sqrt{n})^n \prod_{i=1}^{n} x_i$$

sujeto a:

$$h_1(\vec{x}) = \sum_{i=1}^{n} x_i^2 - 1 = 0$$

donde: $n = 10$ y $0 \leq x_i \leq 1$ ($i = 1, \ldots, n$).

g04.

Minimizar:

$$f(\vec{x}) = 5.3578547x_3^2 + 0.8356891x_1x_5 + 37.293239x_1 - 40792.141$$

sujeto a:

$$\begin{aligned}
g_1(\vec{x}) &= 85.334407 + 0.0056858x_2x_5 + 0.0006262x_1x_4 \\
&\quad -0.0022053x_3x_5 - 92 \leq 0 \\
g_2(\vec{x}) &= -85.334407 - 0.0056858x_2x_5 - 0.0006262x_1x_4 \\
&\quad +0.0022053x_3x_5 \leq 0 \\
g_3(\vec{x}) &= 80.51249 + 0.0071317x_2x_5 + 0.0029955x_1x_2 \\
&\quad +0.0021813x_3^2 - 110 \leq 0 \\
g_4(\vec{x}) &= -80.51249 - 0.0071317x_2x_5 - 0.0029955x_1x_2 \\
&\quad -0.0021813x_3^2 + 90 \leq 0 \\
g_5(\vec{x}) &= 9.300961 + 0.0047026x_3x_5 + 0.0012547x_1x_3 \\
&\quad +0.0019085x_3x_4 - 25 \leq 0 \\
g_6(\vec{x}) &= -9.300961 - 0.0047026x_3x_5 - 0.0012547x_1x_3 \\
&\quad -0.0019085x_3x_4 + 20 \leq 0
\end{aligned}$$

donde: $78 \leq x_1 \leq 102$, $33 \leq x_2 \leq 45$, $27 \leq x_i \leq 45$ ($i = 3, 4, 5$).

g05.

Minimizar:

$$f(\vec{x}) = 3x_1 + 0.000001x_1^3 + 2x_2 + (0.000002/3)x_2^3$$

sujeto a:

$$\begin{aligned} g_1(\vec{x}) &= -x_4 + x_3 - 0.55 \leq 0 \\ g_2(\vec{x}) &= -x_3 + x_4 - 0.55 \leq 0 \\ h_3(\vec{x}) &= 1000\sin(-x_3 - 0.25) + 1000\sin(-x_4 - 0.25) \\ &\quad +894.8 - x_1 = 0 \\ h_4(\vec{x}) &= 1000\sin(x_3 - 0.25) + 1000\sin(x_3 - x_4 - 0.25) \\ &\quad +894.8 - x_2 = 0 \\ h_5(\vec{x}) &= 1000\sin(x_4 - 0.25) + 1000\sin(x_4 - x_3 - 0.25) \\ &\quad +1294.8 = 0 \end{aligned}$$

donde: $0 \leq x_1 \leq 1200$, $0 \leq x_2 \leq 1200$, $-0.55 \leq x_3 \leq 0.55$ y $-0.55 \leq x_4 \leq 0.55$.

g06.

Minimizar:

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

sujeto a:

$$\begin{aligned} g_1(\vec{x}) &= -(x_1 - 5)^2 - (x_2 - 5)^2 + 100 \leq 0 \\ g_2(\vec{x}) &= (x_1 - 6)^2 + (x_2 - 5) - 82.81 \leq 0 \end{aligned}$$

donde: $13 \leq x_1 \leq 100$ y $0 \leq x_2 \leq 100$.

g07.

Minimizar:

$$\begin{aligned} f(\vec{x}) &= x_1^2 + x_2^2 + x_1x_2 - 14x_1 - 16x_2 + (x_3 - 10)^2 \\ &\quad +4(x_4 - 5)^2 + (x_5 - 3)^2 + 2(x_6 - 1)^2 + 5x_7^2 \\ &\quad +7(x_8 - 11)^2 + 2(x_9 - 10)^2 + (x_{10} - 7)^2 + 45 \end{aligned}$$

sujeto a:

$$\begin{aligned} g_1(\vec{x}) &= -105 + 4x_1 + 5x_2 - 3x_7 + 9x_8 \leq 0 \\ g_2(\vec{x}) &= 10x_1 - 8x_2 - 17x_7 + 2x_8 \leq 0 \end{aligned}$$

$$
\begin{aligned}
g_3(\vec{x}) &= -8x_1 + 2x_2 + 5x_9 - 2x_{10} - 12 \leq 0 \\
g_4(\vec{x}) &= 3(x_1 - 2)^2 + 4(x_2 - 3)^2 + 2x_3^2 - 7x_4 - 120 \leq 0 \\
g_5(\vec{x}) &= 5x_1^2 + 8x_2 + (x_3 - 6)^2 - 2x_4 - 40 \leq 0 \\
g_6(\vec{x}) &= x_1^2 + 2(x_2 - 2)^2 - 2x_1x_2 + 14x_5 - 6x_6 \leq 0 \\
g_7(\vec{x}) &= 0.5(x_1 - 8)^2 + 2(x_2 - 4)^2 + 3x_5^2 - x_6 - 30 \leq 0 \\
g_8(\vec{x}) &= -3x_1 + 6x_2 + 12(x_9 - 8)^2 - 7x_{10} \leq 0
\end{aligned}
$$

donde: $-10 \leq x_i \leq 10$ ($i = 1, \ldots, 10$).

g08.

Maximizar:

$$f(\vec{x}) = \frac{\sin^3(2\pi x_1)\sin(2\pi x_2)}{x_1^3(x_1 + x_2)}$$

sujeto a:

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

donde: $0 \leq x_1 \leq 10$, $0 \leq x_2 \leq 10$.

g09.

Minimizar:

$$
\begin{aligned}
f(\vec{x}) &= (x_1 - 10)^2 + 5(x_2 - 12)^2 + x_3^4 + 3(x_4 - 11)^2 + 10x_5^6 \\
&\quad +7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7
\end{aligned}
$$

sujeto a:

$$
\begin{aligned}
g_1(\vec{x}) &= -127 + 2x_1^2 + 3x_2^4 + x_3 + 4x_4^2 + 5x_5 \leq 0 \\
g_2(\vec{x}) &= -282 + 7x_1 + 3x_2 + 10x_3^2 + x_4 - x_5 \leq 0 \\
g_3(\vec{x}) &= -196 + 23x_1 + x_2^2 + 6x_6^2 - 8x_7 \leq 0 \\
g_4(\vec{x}) &= 4x_1^2 + x_2^2 - 3x_1x_2 + 2x_3^2 + 5x_6 - 11x_7 \leq 0
\end{aligned}
$$

donde: $-10 \leq x_i \leq 10$ ($i = 1, \ldots, 7$).

g10.

Minimizar:

$$f(\vec{x}) = x_1 + x_2 + x_3$$

sujeto a:

$$\begin{aligned}
g_1(\vec{x}) &= -1 + 0.0025(x_4 + x_6) \leq 0 \\
g_2(\vec{x}) &= -1 + 0.0025(x_5 + x_7 - x_4) \leq 0 \\
g_3(\vec{x}) &= -1 + 0.01(x_8 - x_5) \leq 0 \\
g_4(\vec{x}) &= -x_1 x_6 + 833.33252 x_4 + 100 x_1 - 83333.333 \leq 0 \\
g_5(\vec{x}) &= -x_2 x_7 + 1250 x_5 + x_2 x_4 - 1250 x_4 \leq 0 \\
g_6(\vec{x}) &= -x_3 x_8 + 1250000 + x_3 x_5 - 2500 x_5 \leq 0
\end{aligned}$$

donde: $100 \leq x_1 \leq 10000$, $1000 \leq x_i \leq 10000$ ($i = 2, 3$) y $10 \leq x_i \leq 1000$ ($i = 4, \ldots, 8$).

g11.

Minimizar:

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

sujeto a:

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

donde: $-1 \leq x_1 \leq 1$, $-1 \leq x_2 \leq 1$.

g12.

Maximizar:

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

sujeto a:

$$g(\vec{x}) = (x_1 - p)^2 + (x_2 - q)^2 + (x_3 - r)^2 - 0.0625 \leq 0$$

donde: $0 \leq x_i \leq 10$ ($i = 1, 2, 3$), y $p, q, r = 1, 2, \ldots, 9$.

g13.

Minimizar:

$$f(\vec{x}) = e^{x_1 x_2 x_3 x_4 x_5}$$

sujeto a:

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

donde: $-2.3 \leq x_i \leq 2.3$ $(i = 1, 2)$ y $-3.2 \leq x_i \leq 3.2$ $(i = 3, 4, 5)$.

En la Tabla 4.1 se muestra la ubicación del óptimo (o el mejor resultado conocido) de cada una de las funciones de prueba, así como algunos comentarios de la función o del óptimo.

Tabla 4.1: Óptimos de las funciones de prueba.

Función de prueba	**Solución óptima** $\vec{x}^*$	$f(\vec{x}^*)$	**Comentarios**
g01	(1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 1)	-15	g_1, g_2, g_3, g_7, g_8 y g_9 están activas en el óptimo.

Tabla 4.1: Óptimos de las funciones de prueba.

Función de prueba	**Solución óptima** $\vec{x}^*$	$f(\vec{x}^*)$	**Comentarios**
g02	(3.16237443645701, 3.12819975856112, 3.09481384891456, 3.06140284777302, 3.02793443337239, 2.99385691314995, 2.95870651588255, 2.92182183591092, 0.49455118612682, 0.48849305858571, 0.48250798063845, 0.47695629293225, 0.47108462715587, 0.46594074852233, 0.46157984137635, 0.45721400967989, 0.45237696886802, 0.44805875597713, 0.44435772435707, 0.44019839654132)	0.803619	g_1 casi está activa en esta solución, que es la mejor conocida.
g03	$\left(\frac{1}{\sqrt{n}}, \ldots, \frac{1}{\sqrt{n}}\right)$	1	Ninguno.
g04	(78, 33, 29.995256025682, 45, 36.775812905788)	-30665.539	g_1 y g_6 están activas en el óptimo.
g05	(679.9453, 1026.067, 0.1188764, -0.3962336)	5126.4981	Esta solución es la mejor conocida.
g06	(14.095, 0.84296)	-6961.8138	Ambas restricciones están activas en el óptimo.

Tabla 4.1: Óptimos de las funciones de prueba.

Función de prueba	**Solución óptima $\vec{x}^*$**	$f(\vec{x}^*)$	**Comentarios**
g07	(2.171996, 2.363683, 8.773926, 5.095984, 0.9906548, 1.430574, 1.321644, 9.828726, 8.280092, 8.375927)	24.3062091	g_1 a g_6 están activas en el óptimo.
g08	(1.2279713, 4.2453733)	0.095825	Esta función tiene muchos óptimos locales, la mayoría cerca del eje x_1.
g09	(2.330499, 1.951372, -0.4775414, 4.365726, -0.6244870, 1.038131, 1.594227)	680.6300573	g_1 y g_4 están activas en el óptimo.
g10	(579.3167, 1359.943, 5110.071, 182.0174, 295.5985, 217.9799, 286.4162, 395.5979)	7049.330	g_1, g_2 y g_3 están activas en el óptimo.
g11	$\left(\pm\frac{1}{\sqrt{2}}, \frac{1}{2}\right)$	0.75	Ninguno.
g12	(5, 5, 5)	1	La región factible está formada por 9^3 esferas. p, q y r pueden tomar cualquier valor entre 0 y 9 para satisfacer la restricción.
g13	(-1.717143, 1.595709, 1.827247, -0.7636413, -0.763645)	0.0539498	Ninguno.

4.2 Resultados

Para obtener los siguientes resultados se realizaron diez ejecuciones del algoritmo por cada función de prueba. Los parámetros en todos los casos son los siguientes:

$$\begin{aligned} p &= 20 \\ G_{máx} &= 2500 \\ g_{normativa} &= 20 \\ aceptación &= 10\% \\ t &= n \\ profundidad_{máx} &= 5 \\ c &= \frac{p}{2} = 10 \end{aligned}$$

El parámetro $profundidad_{máx}$ determina el máximo número de nodos que habrá en el 2^m-*tree* durante la ejecución. Un octree tiene exactamente ocho hijos en un nodo que no es hoja, y con $profundidad_{máx} = 5$, el número máximo de nodos en el árbol será $8^0 + 8^1 + 8^2 + 8^3 + 8^4 = 4681$, aunque no siempre se alcanza esta cantidad. Para comparar con una estructura estática de nodos, diremos que para dividir un espacio con 12 variables de decisión, solamente partiendo a la mitad cada dimensión, se necesitan $2^{12} = 4096$ nodos, lo que es una cantidad comparable. Sin embargo, para un problema con 20 variables (como g02) dividiendo solamente a la mitad cada dimensión, se necesitan $2^{20} = 1,048,576$ nodos.

En las funciones de prueba que tienen restricciones de igualdad (g03, g05, g11 y g13) se utilizó un parámetro extra, δ, de tolerancia para la igualdad. Al introducir este parámetro, todas las restricciones de la forma:

$$h_j(\vec{x}) = 0$$

se transformaron en restricciones de desigualdad de la forma:

$$|h_j(\vec{x})| - \delta \leq 0$$

Este parámetro se ajustó proporcionalmente a las unidades de las variables de decisión para cada problema, considerando los resultados obtenidos durante la experimentación, utilizándose finalmente los siguientes

Tabla 4.2: Resultados de la técnica para manejo de restricciones propuesta en esta tesis.

	Óptimo	**Mejor**	**Media**	**Peor**	**Desv. est.**
g01	-15	-15.0000	-14.4999	-12.0000	1.0801
g02	0.803619	0.747382	0.645249	0.479158	0.076741
g03	1	0.3274	0.0351	0.0000	0.1030
g04	-30665.539	-30665.539	-30662.470	-30636.154	9.251
g05	5126.4981	5239.337	5607.949	6227.275	269.469
g06	-6961.8138	—	—	—	—
g07	24.3062091	60.923	98.130	135.190	25.434
g08	0.095825	0.095825	0.095825	0.095825	0.000000
g09	680.6300573	680.663	681.468	685.180	1.538
g10	7049.330	—	—	—	—
g11	0.75	0.749250*	0.772990	0.859098	0.035318
g12	1	1.000000	0.996375	0.969375	0.009650
g13	0.0539498	0.044440*	0.533874	0.940019	0.346872

valores: para el problema g03, $\delta = 0.05$; para el problema g05, $\delta = 60$; para el problema g11, $\delta = 0.00075$; y para el problema g13, $\delta = 1$.

El número máximo de generaciones se ajustó de forma que se obtuviera un número reducido de evaluaciones de la función objetivo. El total de evaluaciones de la función objetivo durante una ejecución se obtiene calculando $p(G_{máx} + 1)$. Con estos parámetros, en cada ejecución se realizan 50,020 evaluaciones de la función objetivo. En contraste, los resultados reportados para los mapas homomorfos [81] requieren 1,400,000 evaluaciones de la función objetivo, y los resultados reportados para la jerarquización estocástica [123] requieren 350,000 evaluaciones de la función objetivo.

En la Tabla 4.2 se muestran los resultados de la técnica propuesta para optimización con restricciones. Para realizar una comparación, los resultados reportados para la jerarquización estocástica [123] se muestran en la Tabla 4.3, y los resultados reportados para los mapas homomorfos [81] se muestran en la Tabla 4.4.

Los resultados marcados con * son mejores que el óptimo global debido al valor del parámetro δ de tolerancia para las restricciones de igualdad. Cuando en cierto problema alguna de las técnicas no encontró soluciones

Tabla 4.3: Resultados reportados para la jerarquización estocástica

	Óptimo	**Mejor**	**Media**	**Peor**	**Desv. est.**
g01	-15	-15.000	-15.000	-15.000	0.0
g02	0.803619	0.803515	0.781975	0.726288	0.020
g03	1	1.000	1.000	1.000	0.00019
g04	-30665.539	-30665.539	-30665.539	-30665.539	0.00002
g05	5126.4981	5126.497	5128.881	5142.472	3.5
g06	-6961.8138	-6961.814	-6875.940	-6350.262	160
g07	24.3062091	24.307	24.374	24.642	0.066
g08	0.095825	0.095825	0.095825	0.095825	0.000000
g09	680.6300573	680.630	680.656	680.763	0.034
g10	7049.330	7054.316	7559.192	8835.655	530
g11	0.75	0.750	0.750	0.750	0.00008
g12	1	1.000000	1.000000	1.000000	0.0
g13	0.0539498	0.053957	0.057006	0.216915	0.031

Tabla 4.4: Resultados reportados para los mapas homomorfos

Función	**Óptimo**	**Mejor**	**Media**	**Peor**
g01	-15	-14.7864	-14.7082	-14.6154
g02	0.803619	0.79953	0.79671	0.79119
g03	1	0.9997	0.9989	0.9978
g04	-30665.539	-30664.5	-30655.3	-30645.9
g05	5126.4981	—	—	—
g06	-6961.8138	-6952.1	-6342.6	-5473.9
g07	24.3062091	24.620	24.826	25.069
g08	0.095825	-0.095825	-0.089157	-0.029144
g09	680.6300573	680.91	681.16	683.18
g10	7049.330	7147.9	8163.6	9659.3
g11	0.75	0.75	0.75	0.75
g12	1	0.999999	0.999135	0.991950
g13	0.0539498	0.054	0.064	0.557

Tabla 4.5: Razón ρ, entre el tamaño de la zona factible y el tamaño del espacio de búsqueda.

Función de prueba	ρ
g01	0.000235%
g02	99.996503%
g03	0.0000%
g04	26.962511%
g05	0.0000%
g06	0.006679%
g07	0.000103%
g08	0.859082%
g09	0.524450%
g10	0.000522%
g11	0.000000%
g12	4.775265%
g13	0.000000%

factibles, las casillas fueron llenadas con —.

Como puede verse, en la mayoría de las funciones se obtuvieron resultados comparables con las otras técnicas mostradas en la tabla, a pesar del reducido número de evaluaciones de la función objetivo utilizado. Por ejemplo, en la función g08, es muy difícil obtener una ejecución del algoritmo que no alcance el óptimo; en las diez ejecuciones que se utilizaron para formar la Tabla 4.2 siempre se alcanzó el óptimo para esta función.

Sólo en los problemas g06 y g10 no se encontraron individuos factibles durante las ejecuciones, y en los problemas g03 y g07 los resultados fueron notablemente inferiores a las otras técnicas. Esto puede deberse al tamaño de la región factible que, como puede verse en la Tabla 4.5, es muy pequeña comparada con el tamaño del espacio de búsqueda completo. Esto significa que sus regiones factibles son muy pequeñas comparadas con el tamaño de todo el espacio de búsqueda, por lo que encontrar un individuo factible (¡ya no digamos el óptimo!) es una tarea sumamente difícil.

Los valores mostrados en la Tabla 4.5 son una estimación, obtenida generando puntos uniformemente distribuidos en el espacio de búsqueda.

Capítulo 5

Trabajo Previo en Optimización Multiobjetivo

5.1 Optimización Multiobjetivo

El planteamiento del problema de optimización multiobjetivo es similar al de optimización con un solo objetivo. El problema consiste en encontrar el vector solución $\vec{x}$ que optimice:

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

donde $\vec{f}$ es el vector de funciones objetivo, compuesto por las funciones $f_i(\vec{x})$, con $i = 1, 2, \ldots, k$. El problema puede o no tener restricciones.

Sin embargo, el concepto de optimizar varias funciones simultáneamente no es tan simple como encontrar un óptimo para cada función, debido a que las funciones suelen estar en conflicto entre sí. Encontrar el óptimo, entonces, puede interpretarse como encontrar un buen compromiso entre todas las funciones objetivo del problema.

Esta manera de entender el óptimo en un problema multiobjetivo es ambigua, puesto que distintas personas podrían afirmar que puntos distintos representan un buen compromiso, sin que por ello sean óptimos. Vilfredo Pareto [104] dio, hacia finales del siglo XIX, una definición más formal de un óptimo en problemas multiobjetivo, el cual es conocido ahora como el óptimo de Pareto. Esta definición se basa en el concepto de dominancia que es el siguiente. Un punto $\vec{x}$ domina a otro $\vec{y}$ si y sólo si:

$$f_i(\vec{x}) \leq f_i(\vec{y})$$

para $i = 1, 2, \ldots, k$, y que, para al menos un i:

$$f_i(\vec{x}) < f_i(\vec{y})$$

La siguiente definición dice que un punto $\vec{x}^*$ es un óptimo de Pareto, si no existe ningún otro punto $\vec{x}$ que lo domine. Es decir, $\vec{x}^*$ es un óptimo de Pareto si no existe ningún otro punto $\vec{x}$ que cumpla:

$$f_i(\vec{x}) \leq f_i(\vec{x}^*)$$

para $i = 1, 2, \ldots, k$, y que, para al menos un i:

$$f_i(\vec{x}) < f_i(\vec{x}^*)$$

Las condiciones para estas definiciones son válidas suponiendo que se está tratando de minimizar a todas las funciones.

La definición indica que un problema dado no necesariamente tiene sólo un punto que sea óptimo de Pareto. Por lo regular se tiene un conjunto de puntos que cumplen con esta definición. Después de que algún proceso de optimización haya encontrado varios puntos óptimos de Pareto, el tomador de decisiones es la persona encargada de elegir la solución final que ha de utilizarse en la práctica.

Además del óptimo de Pareto, es muy común encontrar en la literatura especializada el término "frente de Pareto". El frente de Pareto es el contradominio del conjunto de puntos que forman el óptimo de Pareto. En problemas con espacios continuos, el frente de Pareto suele ser una línea casi continua. Por esta regularidad, y por mostrar claramente los compromisos y los mejores resultados en cada objetivo, se han utilizado mucho las gráficas del frente de Pareto para mostrar el resultado de los problemas multiobjetivo.

5.2 Computación Evolutiva en Problemas Multiobjetivo

Puesto que la mayoría de los métodos y heurísticas de optimización operan sobre un solo individuo a la vez, es necesario aplicarlos varias veces para encontrar el conjunto de óptimos de Pareto de un problema multiobjetivo. En contraste, los métodos de computación evolutiva son poblacionales, lo que implica que en una sola ejecución pueden arrojar un conjunto de puntos óptimos. Esto es una ventaja adicional que los métodos

de computación evolutiva tienen al resolver problemas de optimización multiobjetivo.

Sin embargo, desde las primeras propuestas se hizo evidente que algunas técnicas eran susceptibles a la forma del frente de Pareto. Por ejemplo, las funciones de agregación lineales (que estuvieron entre las primeras propuestas de algoritmos evolutivos multiobjetivo) son incapaces de generar porciones no convexas del frente de Pareto. Técnicas modernas aún tienen problemas con frentes de Pareto desconectados (discontinuos), por ejemplo.

A continuación se revisan algunas de las técnicas evolutivas que se han propuesto para optimización multiobjetivo, divididas según la clasificación de Cohon y Marks [26], la cual ha sido muy aceptada en la comunidad de investigación de operaciones, y que recientemente es utilizada para clasificar algoritmos evolutivos multiobjetivo [24].

5.3 Técnicas *A Priori*

La primera división en esta clasificación son las técnicas *a priori*, llamadas así, porque las preferencias sobre las funciones objetivo son dadas antes de que la técnica comience la búsqueda. Puede haber técnicas *a priori* de varios tipos, como los que se ven a continuación.

5.3.1 Orden Lexicográfico

En esta técnica se ordenan las funciones objetivo según su importancia, y se optimizan en orden comenzando por la más importante, y terminando por la menos importante. Esta técnica sólo es recomendable cuando se conoce perfectamente el orden de importancia de los objetivos, porque el desempeño del algoritmo depende en gran medida de dicho ordenamiento.

Fourman [50], y Kursawe [82], por ejemplo, han propuesto técnicas basadas en orden lexicográfico.

5.3.2 Funciones de Agregación Lineales

Las funciones de agregación son la manera más simple de adaptar una técnica de optimización de un solo objetivo para problemas multiobjetivo,

porque combinan los resultados de las distintas funciones objetivo en un solo valor de aptitud. Este único valor casi siempre se obtiene mediante una combinación lineal de las funciones objetivo, aunque no es la única manera de hacerlo.

Esta combinación lineal se hace asignando una importancia relativa a cada una de las funciones objetivo, en una constante $w_i > 0$, y donde la suma de todos los pesos es 1. La función de aptitud queda así:

$$aptitud\,(\vec{x}) = \sum_{i=1}^{k} w_i f_i\,(\vec{x})$$

En la mayoría de los casos es necesario introducir también un factor de escalamiento, debido a la heterogeneidad de las unidades de las funciones objetivo.

Entre las aplicaciones que se han reportado de combinaciones lineales están [132, 68, 87, 141].

Implementar un algoritmo que utilice una combinación lineal puede ser muy simple, y también es eficiente, pero una función lineal es incapaz de generar trozos no convexos del frente de Pareto [29]. También suele resultar difícil en la práctica efectuar una buena asignación de pesos para las funciones objetivo.

5.3.3 Funciones de Agregación No Lineales

Aunque sí existen varias propuestas y aplicaciones de funciones de agregación no lineales (por ejemplo, funciones multiplicativas), éstas no han sido muy populares. Tal vez las más utilizadas han sido las técnicas con vectores de metas (p. ej., min-max [73, 128, 101, 133, 59, 25] y programación de metas [11, 66, 140, 124]).

En las técnicas con vectores de metas se trata de minimizar la diferencia entre los valores generados por el proceso de búsqueda, y un vector de objetivos deseados. Estas técnicas pueden generar, bajo ciertas circunstancias, frentes de Pareto no convexos, además de ser bastante eficientes computacionalmente hablando.

Sin embargo, puede ser muy difícil proveer un buen vector de metas, un problema común en todas las técnicas *a priori*.

5.4 Técnicas Progresivas

En este género de técnicas, las preferencias se expresan mientras el proceso está generando soluciones. El tomador de decisiones expresará qué tan buena es una solución para él, y el proceso debe actualizar las preferencias para reflejar las opiniones del tomador de decisiones, antes de continuar con el proceso de búsqueda.

Aunque existen varias técnicas de este tipo en investigación de operaciones, en computación evolutiva prácticamente no se han reportado, si bien varias técnicas exhiben la posibilidad de incorporar las preferencias durante la búsqueda.

5.5 Técnicas *A Posteriori*

Finalmente, se encuentra el conjunto de técnicas en las que las preferencias se expresan al final del proceso de generación de soluciones, y es una actividad que prácticamente no interfiere en la búsqueda. Aquí se intenta generar soluciones que representen todos los compromisos entre las funciones objetivo (es decir, se intenta generar un frente de Pareto completo), y el tomador de decisiones expresará las preferencias al elegir la solución final.

5.5.1 Técnicas de Muestreo Independiente

El muestreo independiente consiste en hacer varias ejecuciones de la técnica para encontrar distintos puntos del frente de Pareto.

Las técnicas de muestreo independiente que se han propuesto en computación evolutiva consisten en funciones agregativas, pero en este caso los pesos no representan las preferencias, sino que son variados para realizar ejecuciones independientes, y así encontrar distintas porciones del frente de Pareto [97, 10, 134, 69, 106].

Estas técnicas son eficientes, y pueden generan buenos frentes si éstos son convexos, pero son aplicables únicamente con un número reducido de funciones objetivo, puesto que el número de combinaciones de pesos por probar crece exponencialmente con el número de objetivos.

5.5.2 Técnicas de Selección por Criterio

La selección divide a la población, y en cada sub-población se realiza la selección basada únicamente en uno de los objetivos. El ejemplo más claro de estas técnicas es el algoritmo de Schaffer llamado VEGA (*Vector Evaluated Genetic Algorithm*) [125].

En VEGA, la población total es dividida en k partes iguales (suponiendo que el problema tiene k objetivos), y en cada una de ellas opera la selección tomando en cuenta sólo una función objetivo. Una vez realizada la selección, la población se mezcla para aplicar el resto de los operadores evolutivos. Todo este proceso se repite en cada generación.

Un problema evidente de VEGA es que no favorece a los buenos compromisos, sino que prefiere las soluciones mejores en sólo uno de los objetivos. A este problema se le conoce como especiación (por su análogo en genética). Este problema fue identificado y atacado por Schaffer, utilizando restricciones a la cruza, al no permitir recombinación entre individuos de la misma sub-población, así como otras heurísticas adicionales durante la selección.

También se mostró [120] que si se utiliza selección proporcional, el esquema de VEGA equivale a una combinación lineal de las funciones objetivo, lo que significa también que comparten limitaciones.

5.5.3 Técnicas de Selección Agregativa

Nuevamente aparecen las funciones agregativas para resolver problemas multiobjetivo. La diferencia esta vez, es que los pesos varían a través de las generaciones o de una evaluación a otra, a lo largo de la ejecución de la técnica [67, 88]. La variación de los pesos se puede hacer de diversas formas, incluso codificándolos en los cromosomas de los individuos.

Sin embargo, nuevamente tenemos la limitante de que, si se utiliza una combinación lineal, es imposible generar el frente de Pareto completo en todos los casos.

5.5.4 Técnicas con Muestreo de Pareto

En este grupo de técnicas se identifican a los individuos no dominados, y se utiliza esa información para hacer la asignación de la aptitud, y realizar la selección.

Existen varias formas de llevar a cabo la selección utilizando muestreo de Pareto, pero la idea original es de Goldberg [54]. Propuso hacer una jerarquización de la siguiente manera: primero se identifican los individuos no dominados de la población, a los que se les asigna la jerarquía más alta, y son ignorados en el resto de la jerarquización. El proceso continúa identificando los no dominados de la población restante, a los que se les asigna el siguiente nivel de jerarquía. Este proceso de jerarquización por capas se realiza hasta que se ha asignado una jerarquía a cada individuo de la población.

También Goldberg sugirió el uso de una técnica de nichos [36], para evitar que la búsqueda evolutiva converja hacia un solo punto no dominado. A partir de la sugerencia de Goldberg, el uso de técnicas para mantener diversidad se ha convertido en una característica general de los algoritmos evolutivos para optimización multiobjetivo, como veremos en los ejemplos mencionados más adelante, pero también implica un trabajo computacional considerable.

El algoritmo llamado MOGA (*Multi-Objective Genetic Algorithm*) de Fonseca y Fleming [48], utiliza otra de las técnicas de jerarquización. En esta propuesta, la jerarquía de un individuo depende del número de individuos en la población actual que lo dominan, como lo muestra la siguiente expresión:

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

donde $p_i^{(t)}$ es el número de individuos que dominan a x_i en la generación t. Como se ve, los individuos no dominados tendrán jerarquía 1. A continuación de la jerarquización, se asignan aptitudes por medio de una interpolación de las jerarquías. Fonseca y Fleming también utilizan una técnica de nichos para evitar la convergencia prematura, pero en este caso sugieren compartir la aptitud en el espacio de las funciones objetivo, y no en el espacio de las variables de decisión.

Srinivas y Deb [129] propusieron el NSGA (*Nondominated Sorting Genetic Algorithm*) que utiliza casi sin cambios la idea de la jerarquización de Goldberg, la cual se hace por capas, que ellos llaman ondas (*waves*). También se comparte aptitud entre los individuos cercanos para mantener diversidad. Una segunda versión de este algoritmo, el NSGA-II, fue propuesta recientemente por Deb et al. [37]. El NSGA-II incorpora elitismo, y un operador para mantener diversidad que no requiere parámetros; es computacionalmente más eficiente que el NSGA, aunque parece que sigue

teniendo problemas para generar regiones aisladas del frente de Pareto, así como para funcionar adecuadamente con representación binaria [24].

A diferencia de los algoritmos antes mencionados, que jerarquizan la población completa, Horn y Nafpliotis [64] propusieron una forma de realizar torneos basados en dominancia de Pareto, con la finalidad de evitar jerarquizar a toda la población. Su propuesta se denomina NPGA (*Niched-Pareto Genetic Algorithm*). En el NPGA, cuando se enfrentan dos individuos en un torneo, se comparan ambos contra una fracción de la población (Horn y Nafpliotis utilizan el 10%) y, si uno resulta dominado y el otro no, gana el no dominado; pero si los dos son dominados o los dos son no dominados, entonces se realiza un conteo de nichos, y gana el individuo que esté en una región menos poblada.

Posteriormente Erickson et al. [43] realizaron una segunda versión de este algoritmo, el NPGA 2, en el que se jerarquiza toda la población antes de realizar los torneos. La segunda diferencia importante es que el conteo de nichos se hace en la población de la siguiente generación, aunque se encuentre incompleta, mientras que en el NPGA se hacía en la población de la generación actual. Esta forma de realizar el conteo de nichos en la siguiente generación fue propuesta previamente por Oei et al. [100].

El SPEA (*Strength Pareto Evolutionary Algorithm*) de Zitzler y Thiele [144], es uno de los algoritmos que usan un archivo externo para almacenar las soluciones no dominadas obtenidas a lo largo del proceso evolutivo. A cada uno de los individuos en este archivo externo no dominado se le asigna un valor de fuerza, el cual es proporcional al número de individuos que domina. Luego, la aptitud de cada individuo de la población depende del valor de fuerza de todos los individuos en el archivo externo que lo dominen, y se realiza una serie de torneos binarios.

También de SPEA existe una segunda versión, llamada SPEA2, de Zitzler et al. [143]. Para calcular la aptitud de un individuo en SPEA2, se utiliza la cantidad de individuos que lo dominan, y la cantidad de individuos que son dominados por él. Además, en SPEA2 el método para truncar el archivo externo asegura que siempre conservará los extremos del frente.

Van Veldhuizen y Lamont [137] propusieron la versión multiobjetivo de un algoritmo genético desordenado [31], llamada MOMGA (*Multi-Objective Messy Genetic Algorithm*). En la primera fase, llamada de iniciación, se generan de manera exhaustiva todos los bloques constructores hasta cierto tamaño. La siguiente fase, llamada primordial, realiza la selección por medio de torneo. Por último, la fase yuxtaposicional genera la

población por medio de un operador de recombinación por corte y empalme.

MOMGA-II fue propuesto por Zydallis et al. [145], y es la versión multiobjetivo del algoritmo genético desordenado rápido [55]. La fase de iniciación también genera bloques constructores, pero esta vez es probabilística, con lo que evitan el crecimiento desmesurado de la población al generar todos los bloques constructores. La segunda fase, llamada de filtrado de bloques constructores, consiste en reducir la población mediante un filtrado, de forma que se almacenan los mejores bloques constructores; además, en esta fase se realiza el torneo de selección. La tercera fase es la misma que la de su predecesor.

Selección Basada en Pareto

Dentro de las técnicas con muestreo de Pareto hay técnicas que utilizan selección basada únicamente en jerarquización de Pareto. Por ejemplo, el algoritmo genético termodinámico (TDGA, *Thermodynamical Genetic Algorithm*) propuesto por Kita et al. [77]. Este algoritmo se basa en una idea similar al recocido simulado, y al momento de hacer la selección trata de minimizar la energía libre del sistema, con lo que mantiene diversidad en la población. En la expresión para calcular la energía se incluye una temperatura que es variada según un horario de enfriamiento.

Otro ejemplo es la técnica de Osyczka y Kundu [102], conocida como "método de la distancia", llamada así porque se basa en el teorema de contacto para calcular distancias relativas de los individuos al frente de Pareto. No requiere un método adicional para mantener diversidad.

Selección Basada en Demes y Pareto

Las técnicas dentro de esta clasificación dividen a su población en varias sub-poblaciones, y en cada una de ellas realizan la jerarquización de Pareto. Por esta jerarquización local, se espera que el proceso sea más eficiente. Sin embargo, debe existir un mecanismo de comunicación entre ellas para obtener resultados globales.

Este tipo de esquemas son especialmente propicios para hacerlos paralelos [122].

Selección Basada en Elitismo y Pareto

En optimización evolutiva de un solo objetivo, se ha demostrado que retener al mejor individuo intacto entre generaciones es necesario para asegurar la convergencia. No obstante, en optimización evolutiva multiobjetivo no se conoce claramente el papel del elitismo, pero la experiencia ha mostrado que puede ser importante, y la mayoría de las técnicas modernas lo incluyen. El elitismo en optimización multiobjetivo es más complicado que en el caso mono-objetivo, porque en optimización mono-objetivo puede identificarse claramente al mejor individuo, pero en optimización multiobjetivo todos los individuos no dominados son igualmente buenos. La forma más común de hacer elitismo en optimización evolutiva multiobjetivo es mediante un archivo externo, que almacene a los individuos no dominados para conservarlos entre generaciones.

El elitismo puede ser el medio principal para conformar una población, utilizando un medio secundario (p. ej., un archivo externo) para llenarla totalmente. Cuando éste sea el caso, entenderemos que se trata de una selección basada en elitismo y Pareto. A continuación se mencionan algunos ejemplos.

PAES (*Pareto Archived Evolution Strategy*), propuesto por Knowles y Corne [78], es una estrategia evolutiva $(1 + 1)$ adicionada con un archivo externo en el que se van almacenando los individuos no dominados encontrados a lo largo del proceso evolutivo. Cuando se encuentra un individuo no dominado es comparado con los individuos del archivo externo, y en caso de que nuevamente sea no dominado, puede entrar al mismo archivo. El mecanismo de PAES para mantener diversidad consiste en una rejilla adaptativa, que es computacionalmente más eficiente que los métodos de nichos.

Knowles y Corne también experimentaron con estrategias evolutivas $(1 + \lambda)$ y $(\mu + \lambda)$, pero aseguran que no hay mejoras significativas y sin embargo sí un aumento en el esfuerzo computacional necesario [78].

Corne et al. [28] propusieron un algoritmo, llamado PESA (*Pareto Envelope-based Selection Algorithm*), en el que se tiene una población pequeña principal, y una población secundaria más grande, que corresponde con el archivo externo mencionado antes. Durante cada iteración del algoritmo, se seleccionan aleatoriamente individuos del archivo externo, y de ellos se producen los nuevos individuos que integrarán la población principal; cuando esta población principal se llene, los individuos no dominados se

incorporarán al archivo secundario.

Posteriormente apareció PESA-II, de Corne et al. [27], cuya diferencia principal con PESA es la selección basada en regiones, lo que significa que la selección considera regiones y no individuos. En una fase siguiente se elige al azar un individuo de cada región seleccionada.

El micro-GA (*micro-Genetic Algorithm*), propuesto por Coello y Toscano [22, 23], es otro algoritmo que tiene una población principal pequeña. Se cuenta con un archivo llamado memoria de la población, y está dividido en una parte reemplazable y una parte no reemplazable, de la cual se eligen aleatoriamente los individuos que formarán la población principal. Esta población pequeña es la que utiliza el algoritmo genético con operadores normales, que cuando converge proporciona los individuos no dominados que entrarán en la memoria externa (archivo externo). Algunos de los individuos en esta memoria externa entrarán periódicamente en la parte reemplazable de la memoria de la población. En total, el micro-GA utiliza tres tipos de elitismo.

5.5.5 Selección Híbrida

Estas técnicas utilizan varios tipos de selección. Comúnmente intentan mezclar dos o más algoritmos, y heredan sus métodos de selección. Estos tipos de selección pueden alternarse durante la ejecución del algoritmo.

Algunas propuestas de selección híbrida son [139, 142, 15, 52].

Para finalizar este capítulo cabe aclarar que hasta el momento de escribir esta tesis no se han reportado otras propuestas que utilicen un algoritmo cultural para optimización multiobjetivo, empleando jerarquización de Pareto.

Capítulo 6

Técnica Propuesta para Optimización Multiobjetivo

El objetivo principal en esta parte del trabajo de tesis es realizar un algoritmo cultural para optimización multiobjetivo que utilice jerarquización de Pareto y elitismo, motivados por la ausencia en la literatura de una propuesta similar [24].

La técnica para optimización multiobjetivo que se presenta aquí, al igual que la técnica para optimización con restricciones, es un CAEP, cuyo pseudo-código se muestra nuevamente en el Algoritmo 7 (levemente modificado con respecto a los anteriores para incluir el uso de un archivo externo, que se describirá más adelante).

La población consiste de un conjunto de individuos, cada uno de los cuales representa una posible solución al problema. Cada individuo contiene las n variables de decisión, las cuales no están codificadas, por tratarse de un algoritmo basado en la programación evolutiva.

La población es iniciada con p individuos generados aleatoriamente, mediante una distribución uniforme dentro de los intervalos para cada variable dados como datos de entrada.

La forma más común de incluir elitismo en un algoritmo evolutivo multiobjetivo es el uso de un archivo externo. El archivo externo es una población secundaria donde se almacenan los individuos no dominados hallados a lo largo del proceso evolutivo. Este archivo tiene la finalidad de conservar las soluciones no dominadas encontradas a lo largo del proceso evolutivo, que podrían ser destruidas por los operadores evolutivos, así como asegurar que se almacenen únicamente soluciones no dominadas

Algoritmo 7 Estructura básica del CAEP multiobjetivo

Generar la población inicial de tamaño p

Evaluar la población inicial

Iniciar el espacio de creencias

Repetir

> Aplicar el operador de mutación bajo la influencia del espacio de creencias para generar p hijos (ahora hay $2p$ hijos en la población)
>
> Evaluar cada hijo
>
> Realizar los torneos binarios, eligiendo aleatoriamente c contrincantes para cada individuo
>
> Seleccionar los p individuos con mayor número de victorias en los torneos, para formar la población de la siguiente generación
>
> Agregar los nuevos individuos no dominados al archivo externo
>
> Actualizar el espacio de creencias con los individuos aceptados

Mientras no se cumpla la condición de finalización

l_{f1}	u_{f1}	l_{f2}	u_{f2}	$\cdots$	l_{fk}	u_{fk}

Figura 6.1: Parte normativa fenotípica

globales (es decir, con respecto a todas las que ha producido el proceso evolutivo), en vez de soluciones no dominadas locales (o sea, con respecto a la generación actual).

El contenido del archivo externo será el resultado final que se le presentará al tomador de decisiones (el frente de Pareto final). Este archivo externo tiene un tamaño máximo q, que corresponde con el número de soluciones que el tomador de decisiones desea que se le presenten como salida del algoritmo.

A continuación se detalla la estructura del espacio de creencias, así como el resto de los pasos del algoritmo.

6.1 Estructura del Espacio de Creencias

El espacio de creencias tiene dos partes: la parte normativa fenotípica, y una rejilla. La rejilla se usa para enfatizar la generación de soluciones no dominadas distribuidas uniformemente a lo largo del frente de Pareto (es decir, en el espacio de las funciones objetivo). Esta rejilla es una variación de la propuesta por Knowles y Corne [78].

La parte del conocimiento normativo fenotípico contiene únicamente los límites inferior y superior, l_{fi} y u_{fi}, de los intervalos para cada función objetivo ($i = 1, \ldots, k$) dentro de los cuales se construirá una rejilla, que se usará para ubicar cada solución no dominada en una especie de sistema coordenado, donde los valores de las funciones objetivo se utilizan para localizar cada solución (ver Figura 6.1).

Teniendo esos intervalos, sólo se requiere conocer el número de subintervalos iguales en los que se dividirán cada uno (s_i, con $i = 1, \ldots, k$), para poder construir la rejilla en el espacio fenotípico. Un ejemplo de esta rejilla se muestra en la Figura 6.2.

Como resultado, se tendrán $s_1 s_2 \ldots s_k$ celdas, que serán todas de las mismas dimensiones. Los valores s_i son parámetros de entrada del algoritmo.

Para cada celda se almacena la cuenta de los individuos no dominados del archivo externo que estén dentro de cada una. Esto es útil para

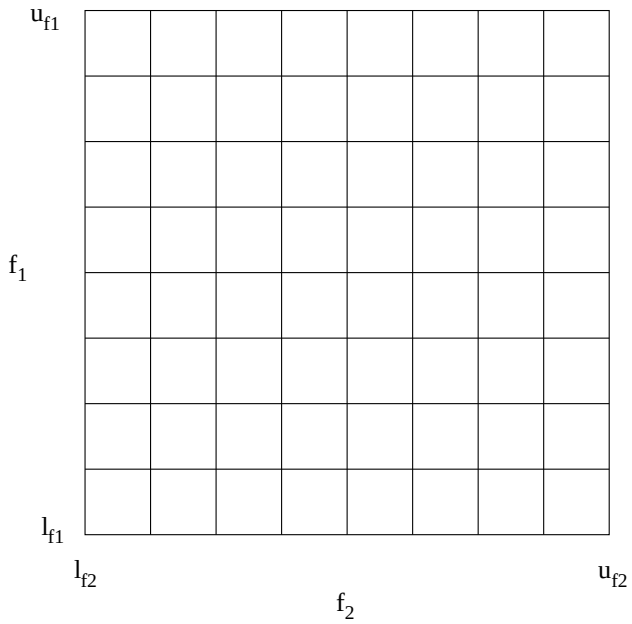

Figura 6.2: Rejilla en el espacio de creencias para un problema con dos funciones objetivo. En este caso, $s_1 = s_2 = 8$ (ocho sub-intervalos en cada dimensión).

distribuir adecuadamente las soluciones no dominadas entre las celdas, evitando que se conglomeren en una zona única del frente de Pareto.

En este caso no se utiliza una estructura de datos como los 2^m-*trees* por la siguiente razón: se requiere que los individuos estén bien distribuidos en el espacio de las funciones objetivo, lo que se intenta llevar a cabo controlando el número de individuos en cada celda. Sin embargo, es fundamental que las celdas sean del mismo tamaño para que este mecanismo tenga éxito; y en los 2^m-trees las celdas son tanto más pequeñas cuantos más individuos tengan dentro, por lo que resultan inadecuados para nuestros fines.

6.2 Iniciación del Espacio de Creencias

Para iniciar el espacio de creencias es necesario que ya exista una población inicial, porque se utilizarán los individuos no dominados de esa población (puede demostrarse que toda población de tamaño diferente de cero contiene al menos un individuo no dominado [135]).

6.2.1 Iniciación de la Parte Normativa Fenotípica

La iniciación de la parte normativa fenotípica del espacio de creencias consiste en encontrar los valores extremos de cada función objetivo que se encuentren en los individuos no dominados de la población inicial. Esos extremos se almacenarán en l_{fi} y u_{fi}, para ubicar la rejilla en la región donde se encuentran los individuos no dominados conocidos hasta el momento.

6.2.2 Iniciación de la Rejilla

La rejilla se crea tomando como intervalos los valores almacenados en la parte normativa fenotípica, y se divide utilizando los parámetros de entrada s_i.

Los contadores de los individuos no dominados dentro de cada celda se inician con cero.

6.3 Actualización del Espacio de Creencias

La rejilla del espacio de creencias se actualiza a cada generación, mientras que la parte normativa fenotípica se actualiza a cada $g_{normativa}$ generaciones, siendo $g_{normativa}$ un parámetro de entrada del algoritmo.

6.3.1 Actualización de la Rejilla

Para actualizar la rejilla simplemente se incrementan los contadores de los individuos no dominados con todos los individuos recién agregados al archivo externo durante la generación actual.

La actualización de la rejilla es bastante simple, y esa es la razón por la que se ejecuta a cada generación. Para la actualización de esta parte del espacio de creencias, la función de aceptación utiliza la población del archivo externo, y elige únicamente a los individuos nuevos en esa población.

6.3.2 Actualización de la Parte Normativa Fenotípica

La actualización de la parte normativa fenotípica no se realiza a cada generación, porque implica una reconstrucción de la rejilla. Para su realización también se utiliza la población del archivo externo.

Nuevamente, como en la iniciación de esta parte normativa fenotípica, es necesario identificar los valores extremos en cada función objetivo de los individuos que se encuentran en el archivo externo. Esos valores se almacenan en l_{fi} y u_{fi} para $i = 1, \ldots, k$.

Con los nuevos valores se reconstruye la rejilla, que abarcará todo el frente de Pareto actual (Figura 6.3), y que posiblemente se habría extendido más allá de los límites de la rejilla anterior.

Después de reiniciar la rejilla, todos los contadores estarán en cero, y es necesario agregar todos los individuos del archivo externo al contador de su celda correspondiente, para que el espacio de creencias esté listo nuevamente para su uso.

6.4 Mutación

La información almacenada en el espacio de creencias pertenece al espacio fenotípico de nuestro problema, por lo que es difícil aplicarla a la autoa-

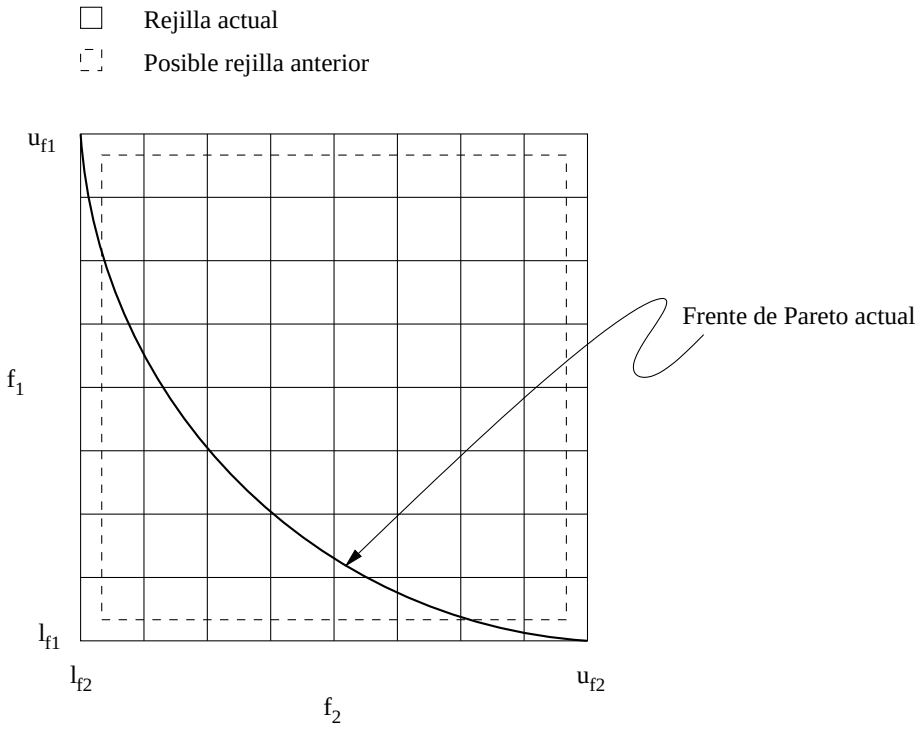

Figura 6.3: La actualización de la parte normativa fenotípica del espacio de creencias tiene como finalidad que todo el frente de Pareto actual esté justo dentro de la rejilla.

daptación de la mutación (que se realiza en el espacio genotípico). Dado este inconveniente, dejamos a los parámetros de la mutación como parámetros de entrada del algoritmo, a ser definidos por el usuario.

La mutación gaussiana para nuestro algoritmo, entonces, sigue la siguiente expresión:

$$x_i' = x_i + N(0, \sigma)$$

donde x_i es la variable i del individuo x, x_i' es la variable i del nuevo individuo x' resultante de la mutación, y $N(\mu, \sigma)$ es una variable aleatoria con una distribución normal con media μ y desviación estándar σ. Para este algoritmo, μ siempre será cero, y σ es un parámetro dado por el usuario.

La mutación se realiza para $i = 1, \ldots, n$, y opera sobre los p individuos de la población principal, por lo que al final de este proceso se tendrá una población de tamaño $2p$.

6.5 Torneo de selección

El torneo de selección es muy similar al de nuestro algoritmo cultural para optimización con restricciones. Se efectúa considerando a la población principal de tamaño $2p$.

Cada individuo se enfrentará contra otros c individuos, elegidos al azar de la población principal. Las reglas para el torneo son:

1. Si un individuo domina al otro, gana el individuo no dominado.
2. Si no son comparables, o sus valores de las funciones objetivo son iguales, entonces:
   (a) Si ambos están dentro de la rejilla del espacio de creencias, gana el que se encuentre en una celda menos poblada (según el contador de las celdas).
   (b) Si alguno cae fuera de la rejilla, gana el que esté fuera.

La primera regla es clara, estamos prefiriendo individuos no dominados para acercarnos al verdadero frente de Pareto. Luego, el primer inciso del segundo punto tiene como finalidad distribuir las soluciones de manera equitativa entre las celdas, y construir un frente de Pareto más uniforme. Finalmente, si se presenta el caso del segundo inciso, significa que hemos

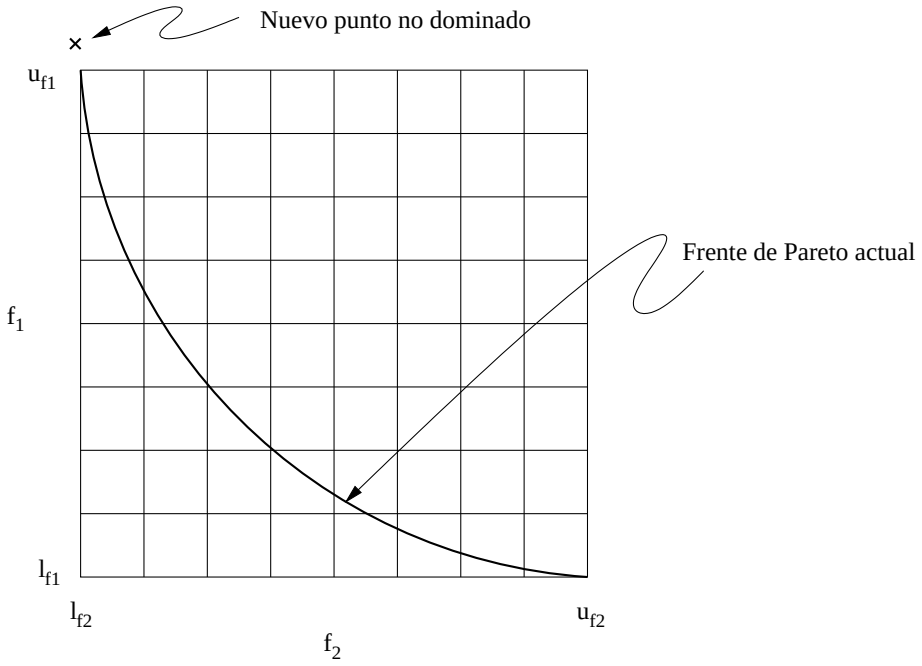

Figura 6.4: Torneo cuando un punto está fuera de la rejilla. El punto encontrado es un nuevo extremo del frente de Pareto actual, y es importante conservarlo.

encontrado una solución que está más allá del frente de Pareto conocido. Dado que esta solución generará una nueva porción del frente de Pareto, es importante conservarla. En la Figura 6.4 se muestra el caso en el que un individuo está fuera de la rejilla.

Una vez concluidos los torneos, se selecciona a los individuos con un número mayor de victorias para pasar a la siguiente generación.

Puede verse que estos torneos son similares a los del NPGA [65], donde se compara contra una fracción de la población. La diferencia es que en el NPGA se jerarquiza contra toda esa fracción de la población, mientras que en este algoritmo la comparación se hace uno a uno, contabilizando el número de victorias (como se hace comúnmente en la programación evolutiva [45]).

6.6 Adición de Individuos al Archivo Externo

El archivo externo debe contener únicamente individuos no dominados, sin repetirlos. Para agregar individuos a este archivo, se tienen las siguientes reglas:

1. Si el individuo que se pretende agregar es dominado por algún individuo del archivo externo, entonces el individuo no se debe agregar.
2. Si el individuo que se pretende agregar domina a algún individuo del archivo externo, entonces se introduce en su lugar, pero continúa comparándose contra todos los demás. Si el mismo individuo, ya agregado, dominara a algún otro, éste (el dominado) es eliminado del archivo externo.
3. Si el individuo que se pretende agregar no es dominado ni domina a ningún otro en el archivo externo, y el tamaño actual de este archivo externo es menor que su tamaño máximo q, entonces se puede agregar el nuevo individuo.
4. Si el individuo que se pretende agregar no es dominado ni domina a ningún otro en el archivo externo, y el tamaño actual de este archivo externo es su tamaño máximo q, entonces se busca algún individuo del archivo externo cuya celda contenga más individuos que la celda a la que pertenece el individuo que se pretende agregar, y se reemplaza el individuo anterior con el nuevo. Con esto se obtiene una mejor distribución de los individuos no dominados entre las celdas.

Aunque el archivo externo esté lleno, su contenido seguirá cambiando durante la ejecución para obtener una mejor distribución. Al final, este archivo externo tendrá el frente de Pareto que se mostrará al tomador de decisiones, quien elegirá una de las soluciones para utilizarla en el problema que se quiere resolver.

Para implementar este archivo externo se utilizó una estructura lineal.

6.7 Parámetros de la Técnica

En la Tabla 6.1 se muestran los parámetros que utiliza esta técnica. Los valores recomendados son de acuerdo con nuestras observaciones, después de numerosos experimentos.

Tabla 6.1: Parámetros de la técnica para optimización multiobjetivo

Parámetro	**Significado**	**Valor recomendado**
p	Tamaño de la población.	Esta técnica tiene un buen funcionamiento con tamaños pequeños de población, y por lo regular no mejora notablemente si se le da un valor grande. Los valores que dan buenos resultados están entre 4 y 20.
q	Tamaño máximo del archivo externo.	Este valor puede ser igual que el número de soluciones que desea el tomador de decisiones como salida. Sin embargo, debe considerarse que si es un número demasiado grande, el algoritmo puede no dar tantas soluciones. Un valor entre 100 y 200 suele ser razonable.
G_{max}	Número máximo de generaciones.	Dependiendo del problema, con 10,000 generaciones se puede llenar y distribuir un archivo externo de 100 individuos.
s_i, con $i = 1, \ldots, k$	Número de subintervalos en los que se dividirá la dimensión i para formar la rejilla.	Con este parámetro se regula la mayor parte del consumo de memoria, puesto que habrá $s_1 s_2 \ldots s_k$ nodos en la rejilla. Si se tienen hasta cuatro funciones objetivo ($k = 4$), 10 es un valor apropiado para todas las s_i.

Tabla 6.1: Parámetros de la técnica para optimización multiobjetivo

Parámetro	**Significado**	**Valor recomendado**
$g_{normativa}$	Frecuencia, en generaciones, con la que se actualiza la parte normativa fenotípica	Un número pequeño aumenta el tiempo de máquina necesario, y un número grande puede hacer que se pierdan los extremos del frente de Pareto y que no se distribuya bien. Un valor intermedio sugerido es 20.
c	Número de encuentros por individuo durante el torneo.	Entre más cercano sea este número a $2p$, más parecido será a una jerarquización completa, requiriendo, en consecuencia, más tiempo de máquina. $\frac{p}{2}$ es un buen valor.
σ	Desviación estándar para la mutación.	Este valor depende de las unidades de las variables de decisión del problema. En nuestros experimentos, $\sigma = 1$ siempre dio buenos resultados.

Capítulo 7

Comparación de Resultados en Optimización Multiobjetivo

7.1 Funciones de Prueba

Para validar la técnica propuesta en esta tesis, se utilizó un conjunto de 7 funciones de prueba, sugerido por Coello en [24]. La primera función, MOP1, fue propuesta por Schaffer [125]; MOP2 es el segundo problema multiobjetivo de Fonseca [49]; el tercer problema fue propuesto por Poloni en [107]; MOP4 fue propuesto por Kursawe en [82]; MOP5 es uno de los problemas multiobjetivo de Viennet propuestos en [138]; el sexto problema fue diseñado siguiendo la metodología de Deb [32, 33, 34]; finalmente, MOP7 es otro de los problemas propuestos por Viennet [138].

Las funciones de prueba se muestran a continuación.

MOP1.

Minimizar:

$$\vec{f}(x) = (f_1(x), f_2(x))$$

donde:

$$\begin{aligned} f_1(x) &= x^2 \\ f_2(x) &= (x-2)^2 \end{aligned}$$

y $-10^5 \leq x \leq 10^5$.

MOP2.

Minimizar:

$$\vec{f}(\vec{x}) = (f_1(\vec{x}), f_2(\vec{x}))$$

donde:

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

y $-4 \leq x_i \leq 4$ $(i = 1, 2, 3)$.

MOP3.

Maximizar:

$$\vec{f}(\vec{x}) = (f_1(x, y), f_2(x, y))$$

donde:

$$\begin{aligned} f_1(x, y) &= -[1 + (A_1 - B_1)^2 + (A_2 - B_2)^2] \\ f_2(x, y) &= -[(x + 3)^2 + (y + 1)^2] \end{aligned}$$

con:

$$\begin{aligned} A_1 &= 0.5\sin 1 - 2\cos 1 + \sin 2 - 1.5\cos 2 \\ A_2 &= 1.5\sin 1 - \cos 1 + 2\sin 2 - 0.5\cos 2 \\ B_1 &= 0.5\sin x - 2\cos x + \sin y - 1.5\cos y \\ B_2 &= 1.5\sin x - \cos x + 2\sin y - 0.5\cos y \end{aligned}$$

y $-\pi \leq x, y \leq \pi$.

MOP4.

Minimizar:

$$\vec{f}(\vec{x}) = (f_1(\vec{x}), f_2(\vec{x}))$$

donde:

$$\begin{aligned} f_1(\vec{x}) &= \sum_{i=1}^{n-1} \left(-10e^{(-0.2)\sqrt{x_i^2+x_{i+1}^2}} \right) \\ f_2(\vec{x}) &= \sum_{i=1}^{n} (|x_i|^a + 5\sin(x_i^b)) \end{aligned}$$

con:

$$\begin{aligned} a &= 0.8 \\ b &= 3 \end{aligned}$$

y $-5 \leq x_i \leq 5$ $(i = 1, 2, 3)$.

MOP5.

Minimizar:

$$\vec{f}(x, y) = (f_1(x, y), f_2(x, y), f_3(x, y))$$

donde:

$$\begin{aligned} f_1(x, y) &= 0.5(x^2 + y^2) + \sin(x^2 + y^2) \\ f_2(x, y) &= \frac{(3x - 2y + 4)^2}{8} + \frac{(x - y + 1)^2}{27} + 15 \\ f_3(x, y) &= \frac{1}{(x^2 + y^2 + 1)} - 1.1e^{(-x^2-y^2)} \end{aligned}$$

y $-30 \leq x, y \leq 30$.

MOP6.

Minimizar:

$$\vec{f}(x, y) = (f_1(x, y), f_2(x, y))$$

donde:

$$\begin{aligned} f_1(x, y) &= x \\ f_2(x, y) &= (1 + 10y) \left[1 - \left(\frac{x}{1 + 10y} \right)^{\alpha} - \frac{x}{1 + 10y} \sin(2\pi q x) \right] \end{aligned}$$

con:

$$\begin{aligned} q &= 4 \\ \alpha &= 2 \end{aligned}$$

y $0 \leq x, y \leq 1$.

MOP7.

Minimizar:

$$\vec{f}(x, y) = (f_1(x, y), f_2(x, y), f_3(x, y))$$

donde:

$$\begin{aligned} f_1(x, y) &= \frac{(x-2)^2}{2} + \frac{(y-1)^2}{13} + 3 \\ f_2(x, y) &= \frac{(x+y-3)^2}{36} + \frac{(-x+y+2)^2}{8} - 17 \\ f_3(x, y) &= \frac{(x+2y-1)^2}{175} + \frac{(2y-x)^2}{17} - 13 \end{aligned}$$

y $-400 \leq x, y \leq 400$.

El problema MOP1 posee un frente de Pareto convexo, su conjunto óptimo está conectado, y es considerado como un problema "fácil", porque existe una solución analítica [136], y porque tiene una sola variable de decisión.

MOP2 también tiene un óptimo conectado, y su frente de Pareto es convexo. Este problema es escalable, porque puede incrementarse el número de variables de decisión (n) sin cambiar la forma ni la ubicación del frente de Pareto [49].

MOP3 tiene una forma más complicada que los primeros dos problemas. Tanto el óptimo como el frente de Pareto consisten en dos áreas desconectadas relativamente lejanas, por lo que en algunos casos es difícil encontrar ambas.

El conjunto óptimo del problema de Kursawe, MOP4, tiene varias regiones desconectadas. Este problema también es escalable en el número de variables de decisión, pero con cada variación la forma del frente de Pareto cambia [24].

El problema MOP5 posee un conjunto óptimo desconectado, mientras que su frente de Pareto es una sola curva.

MOP6 es un problema con un conjunto óptimo y un frente de Pareto desconectados en cuatro regiones.

Por último, el problema MOP7 tiene un conjunto óptimo conectado que forma una superficie.

Las diversas características que presentan estos problemas nos permiten apreciar el funcionamiento del algoritmo bajo distintas condiciones.

7.2 Resultados

Los parámetros que se utilizaron para obtener los resultados que se muestran más adelante son los siguientes:

$$\begin{aligned} p &= 6 \\ q &= 100 \\ G_{máx} &= 35,000 \\ g_{normativa} &= 20 \\ s_i &= 10 \ (i = 1, \ldots, k) \\ c &= \frac{p}{2} = 3 \\ \sigma &= 1 \end{aligned}$$

El parámetro σ es muy importante para obtener un frente de Pareto bien distribuido, y cercano al verdadero frente de Pareto. Sin embargo, curiosamente, este parámetro parece no depender exactamente del tamaño del espacio de búsqueda, sino de los intervalos dentro de los cuales caen todos los puntos que pertenecen al óptimo de Pareto. Para las funciones de prueba que se utilizaron aquí, se logró uniformar este parámetro en 1.

El número de evaluaciones de las funciones objetivo durante una ejecución del algoritmo se calcula con $p(G_{máx} + 1)$. En este caso se llevaron a cabo 210,006 evaluaciones de las funciones objetivo en cada ejecución.

En las Figuras 7.1 a 7.7 se muestran las gráficas que comparan el verdadero frente de Pareto con los resultados de nuestra técnica en las siete funciones de prueba. Los verdaderos frentes de Pareto fueron generados por enumeración, con resolución de 0.01 en cada una de las variables de decisión. Las ejecuciones mostradas fueron realizadas con semillas de números aleatorios generadas al azar, por las mismas funciones utilizadas dentro de los algoritmos.

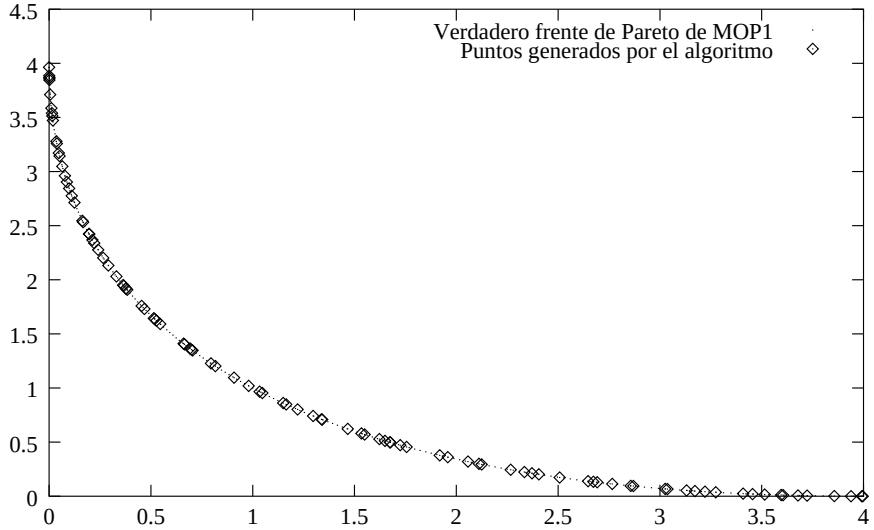

Figura 7.1: Verdadero frente de Pareto y frente generado por el algoritmo en MOP1

Las gráficas anteriores ayudan a evaluar el desempeño del algoritmo de manera visual. Se ve, por ejemplo, que en problemas como MOP1 o MOP6 los puntos generados casi coinciden en su totalidad con el verdadero frente de Pareto, y además se encuentran bien distribuidos. También se puede ver que en MOP4 los puntos generados por el algoritmo en el centro de las curvas quedan un poco alejados del verdadero frente de Pareto.

Para realizar una evaluación cuantitativa de los resultados en optimización evolutiva multiobjetivo, se han ideado métricas. En esta tesis se aplicaron tres métricas: tasa de error, distancia generacional y espaciado.

- La tasa de error (ER) nos dice el porcentaje de individuos que están sobre el verdadero frente de Pareto. Matemáticamente se representa así:

$$ER = \frac{\sum_{i=1}^{n} e_i}{n}$$

donde n es el número de soluciones generadas por el algoritmo y

$$e_i = \begin{cases} 0 & \text{si la solución } \vec{x}_i \text{ pertenece al verdadero frente de Pareto,} \\ 1 & \text{en otro caso} \end{cases}$$

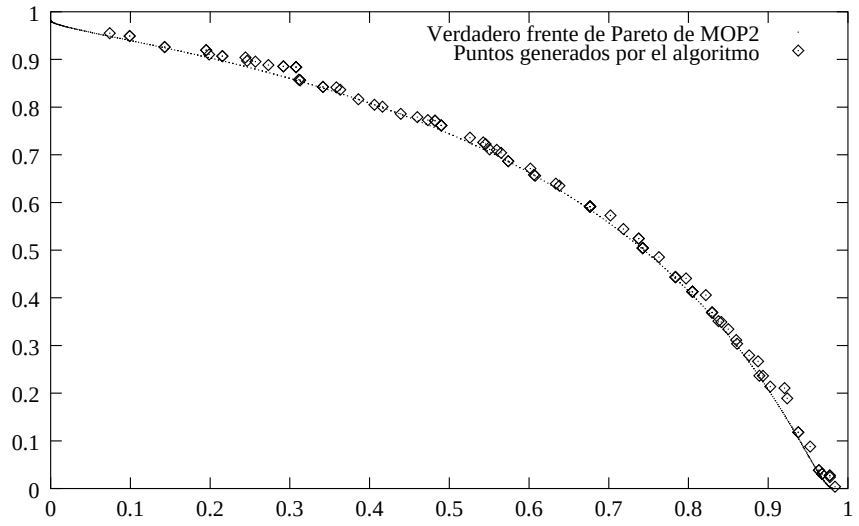

Figura 7.2: Verdadero frente de Pareto y frente generado por el algoritmo en MOP2

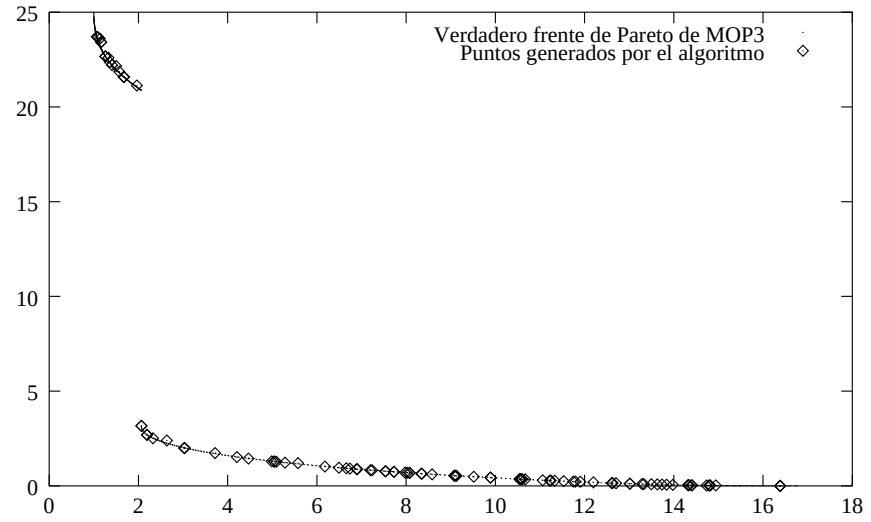

Figura 7.3: Verdadero frente de Pareto y frente generado por el algoritmo en MOP3

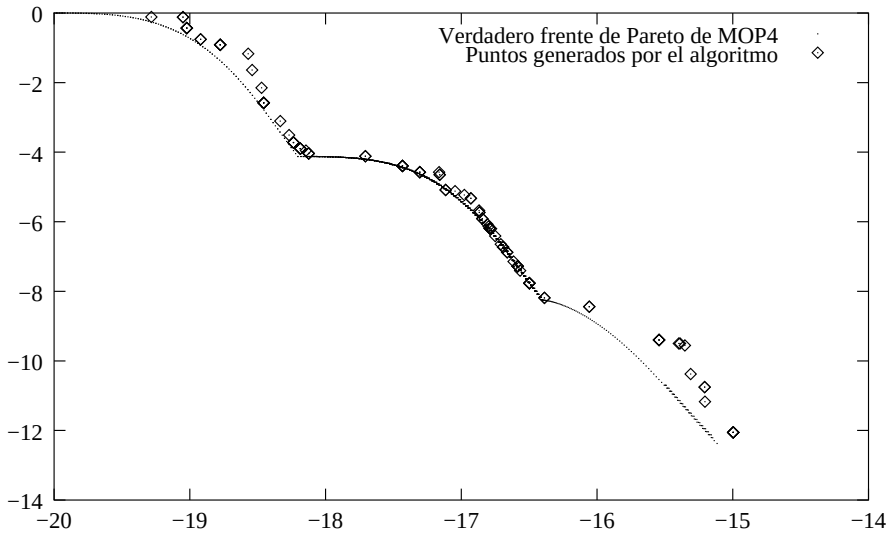

Figura 7.4: Verdadero frente de Pareto y frente generado por el algoritmo en MOP4

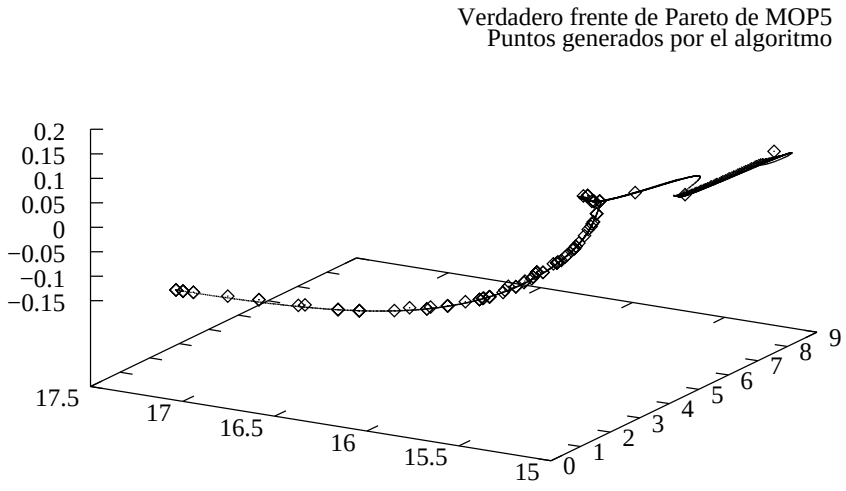

Figura 7.5: Verdadero frente de Pareto y frente generado por el algoritmo en MOP5

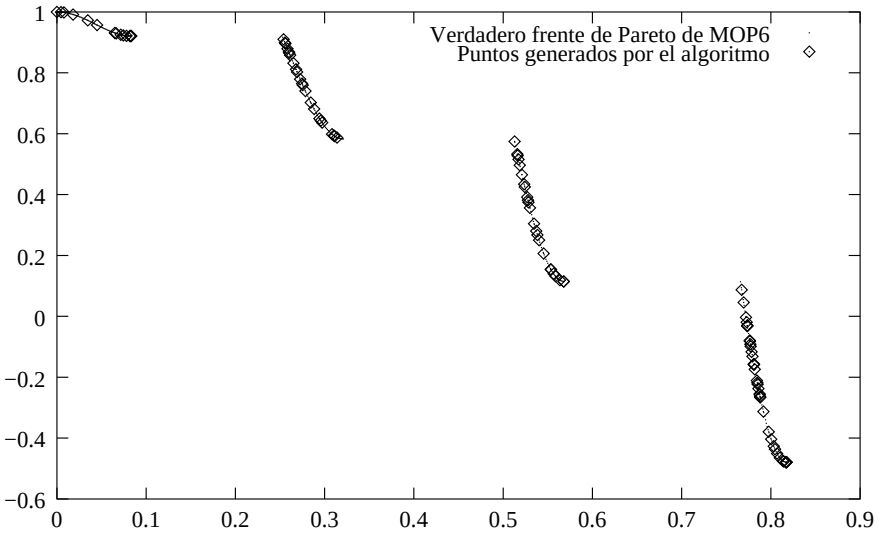

Figura 7.6: Verdadero frente de Pareto y frente generado por el algoritmo en MOP6

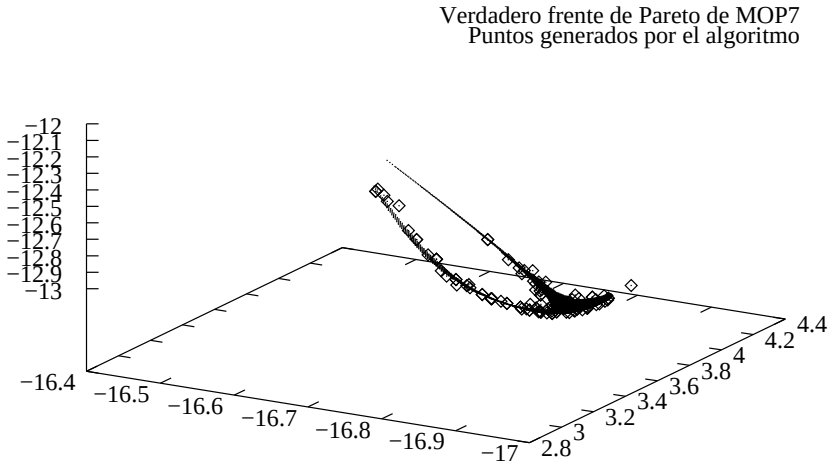

Figura 7.7: Verdadero frente de Pareto y frente generado por el algoritmo en MOP7

con $i = 1, 2, \ldots, n$.

- La distancia generacional (GD) indica qué tan lejos están las soluciones generadas por el algoritmo del verdadero frente de Pareto. Es útil, porque puede darse el caso que algoritmos que presentan tasas de error grandes, generen soluciones muy cercanas al verdadero frente de Pareto. Su expresión es la siguiente:

$$GD = \frac{\left(\sum_{i=1}^{n} d_i^p\right)^{\frac{1}{p}}}{n}$$

  donde n es el número de soluciones generadas por el algoritmo, $p = 2$, y d_i es la distancia euclidiana entre la solución $\vec{x}_i$ y el miembro más cercano del verdadero frente de Pareto, medida en el espacio de las funciones objetivo (con $i = 1, 2, \ldots n$).

- Por último, la métrica de espaciado (S) indica qué tan bien distribuidas están las soluciones. Un valor de cero significa que todas las soluciones están a la misma distancia unas de otras. Su expresión es la siguiente:

$$S = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} \left(\bar{d} - d_i\right)^2}$$

  donde n es el número de soluciones generadas por el algoritmo, $d_i = \min_j \left(|f_1\left(\vec{x}_i\right) - f_1\left(\vec{x}_j\right)| + |f_2\left(\vec{x}_i\right) - f_2\left(\vec{x}_j\right)|\right)$ (con $i, j = 1, 2, \ldots n$), y $\bar{d}$ es la media de todas las d_i.

En la Tabla 7.1 se muestran los resultados del algoritmo que se propone en esta tesis, en términos de las tres métricas mencionadas. Como punto de referencia, también se incluyen los resultados del NSGA-II [37], que es representativo del estado del arte en optimización evolutiva multiobjetivo, aplicado a los mismos problemas. En ambos casos, se realizaron 10 ejecuciones para calcular las estadísticas.

Los parámetros que se utilizaron para las ejecuciones del NSGA-II son los siguientes: tamaño de la población = 100, número máximo de generaciones = 2100, probabilidad de cruza = 0.9, probabilidad de mutación = $\frac{1}{n}$, parámetro SBX = 10 y parámetro de mutación = 100. Los valores para estos parámetros son sugeridos por Deb en el mismo código, excepto el número de generaciones, que se eligió para aproximar el número de evaluaciones

de las funciones objetivo hechas por nuestra técnica; con estos parámetros, el NSGA-II realizó 210,100 evaluaciones de las funciones objetivo en cada ejecución.

Como puede verse en la Tabla 7.1, en el problema MOP1, el algoritmo que se propone en esta tesis mejoró en todas las métricas al NSGA-II. Se confirman también los resultados satisfactorios para MOP3 y MOP6.

En MOP5 se presentaron algunos problemas de convergencia prematura, debido probablemente a la forma sinuosa del verdadero frente de Pareto. Tal vez la principal debilidad de la técnica propuesta sea precisamente la dispersión de las soluciones.

Tabla 7.1: Comparación de resultados entre el algoritmo propuesto en esta tesis (CAEP) y el NSGA-II.

Métrica:	**ER**		**GD**		**S**	
	CAEP	**NSGA-II**	**CAEP**	**NSGA-II**	**CAEP**	**NSGA-II**
MOP1						
Media	0.003	0.008059	0.000946	0.061548	0.037342	0.058298
Desv. est.	0.006749	0.005818	0.000046	0.002711	0.002674	0.009385
Min.	0	0	0.000899	0.058940	0.034057	0.043401
Máx.	0.02	0.017857	0.001045	0.067989	0.042604	0.075381
MOP2						
Media	0.911	0.574	0.001103	0.000280	0.010931	0.007241
Desv. est.	0.046774	0.034059	0.000171	0.000034	0.001040	0.000741
Min.	0.84	0.53	0.000934	0.000230	0.009493	0.006015
Máx.	0.98	0.63	0.001514	0.000349	0.012157	0.008287
MOP3						
Media	0.149	0.209	0.015321	0.001941	0.607229	0.092216
Desv. est.	0.069194	0.041218	0.026480	0.000078	1.025953	0.008415
Min.	0.06	0.15	0.001942	0.001822	0.065285	0.081569
Máx.	0.28	0.28	0.076256	0.002107	2.80319	0.106568
MOP4						
Media	0.876	0.144	0.022522	0.002892	0.139053	0.038378
Desv. est.	0.101017	0.049035	0.008002	0.000203	0.049507	0.003837
Min.	0.64	0.05	0.013604	0.002525	0.094707	0.031393
Máx.	0.97	0.21	0.037278	0.003199	0.242052	0.044254
MOP5						
Media	0.801727	0.590604	61.29437	5.414075	9.431700	0.595304
Desv. Est.	0.176937	0.193508	83.98263	2.218301	11.30065	0.276558
Min.	0.59	0.32	0.001401	3.05536	0.289847	0.276492
Máx.	1	0.85	186.619	8.30579	24.682	1.23193
MOP6						
Media	0.028	0	0.000259	0.000337	0.014476	0.008266
Desv. est.	0.018135	0	0.000028	0.000013	0.005168	0.000918
Min.	0	0	0.000220	0.000318	0.009296	0.006851
Máx.	0.06	0	0.000302	0.000355	0.022920	0.010127
MOP7						
Media	0.528	0.209	0.001607	0.000723	0.024400	0.014575
Desv. est.	0.048717	0.069033	0.000492	0.000674	0.004067	0.009922
Min.	0.48	0.12	0.000659	0.000270	0.020067	0.005501
Máx.	0.64	0.32	0.002361	0.002558	0.031742	0.034569

Conclusiones y Trabajo Futuro

Las técnicas de computación evolutiva han mostrado tener un buen desempeño en una gran variedad de problemas difíciles, pero se siguen explorando alternativas dentro de esta clase de heurísticas.

En esta tesis se exploran las posibilidades de utilizar los algoritmos culturales, que son una técnica evolutiva, sobre dos clases de problemas: la optimización mono-objetivo con restricciones, y la optimización multiobjetivo. La idea central de los algoritmos culturales es incorporar conocimiento del domino adquirido durante el mismo proceso de búsqueda, integrando dentro del algoritmo evolutivo el llamado "espacio de creencias". El objetivo es hacer algoritmos más robustos y con tasas de convergencia mayores, pero para conseguirlo se requiere que el espacio de creencias sea cuidadosamente diseñado para el tipo específico de problemas que se desea resolver, de modo que codifique y almacene adecuadamente las experiencias de la población. Además, se requiere que estas experiencias sean útiles para los nuevos individuos.

En el primer tipo de problemas que se atacaron (los de optimización con restricciones) ya existían propuestas de algoritmos culturales. Por lo tanto, el énfasis en esta parte de la tesis fue construir un algoritmo competitivo. El algoritmo resultante arrojó resultados comparables con las técnicas evolutivas más importantes en optimización con restricciones, e incrementó las tasas de convergencia (lo cual se ve reflejado en el número de evaluaciones realizadas de la función objetivo); todo lo anterior utilizando eficientemente la memoria del sistema en el cual se ejecuta, cosa que era una desventaja importante en sus predecesores.

En el segundo tipo de problemas (los de optimización multiobjetivo) no existía ninguna propuesta de algoritmos culturales que utilizaran jerarquización de Pareto. El énfasis en esta parte de la tesis fue mostrar la factibilidad de dicha propuesta. Se obtuvo un algoritmo cultural para

optimización multiobjetivo que logra aproximar los frentes de Pareto de problemas con diversas características, y que incluso consigue mejorar, en al menos un caso, los resultados de otro algoritmo evolutivo, en condiciones similares (aproximadamente con el mismo número de evaluaciones de las funciones objetivo). Ese otro algoritmo es el NSGA-II, que es representativo del estado del arte en optimización evolutiva multiobjetivo.

En cuanto al trabajo futuro que se puede realizar sobre el algoritmo cultural para optimización con restricciones, está la adición de algún mecanismo para manejar más adecuadamente las restricciones de igualdad, de forma que se reduzca la importancia o se elimine totalmente al parámetro δ. Otra debilidad que se logró identificar en esta técnica es su operación en espacios con regiones factibles muy pequeñas (relativamente al tamaño del espacio de búsqueda completo). En este sentido, un mecanismo para identificar las zonas factibles podría ayudar al desempeño del algoritmo.

Entre las debilidades que presenta el algoritmo cultural para optimización multiobjetivo, está la pérdida de diversidad y la convergencia prematura en algunos problemas. Para solucionar esto, se requiere un mecanismo para incrementar diversidad en la población principal. Otro problema es la dificultad para dispersar uniformemente las soluciones a lo largo del frente de Pareto en algunos problemas. Para esta debilidad existen dos soluciones posibles: una es mejorar la operación de la rejilla, y la otra es sustituir definitivamente la rejilla por otra técnica de dispersión. También es deseable lograr la autoadaptación del operador de mutación con la información almacenada en el espacio de creencias. En cualquier caso, para validar resultados se requiere la aplicación de métricas y la comparación con más algoritmos que hayan mostrado ser competitivos.

Bibliografía

[1] Hojjat Adeli and Nai-Tsang Cheng. Augmented Lagrangian Genetic Algorithm for Structural Optimization. *Journal of Aerospace Engineering*, 7(1):104–118, January 1994.

[2] Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz, editors. *Handbook of Evolutionary Computation*. Institute of Physics Publishing and Oxford University Press, 1997.

[3] James C. Bean. Genetics and random keys for sequencing and optimization. *ORSA Journal on Computing*, 6(2):154–160, 1994.

[4] James C. Bean and Atidel Ben Hadj-Alouane. A Dual Genetic Algorithm for Bounded Integer Programs. Technical Report TR 92-53, Department of Industrial and Operations Engineering, The University of Michigan, 1992. To appear in R.A.I.R.O.-R.O. (invited submission to special issue on GAs and OR).

[5] Sheela V. Belur. CORE: Constrained Optimization by Random Evolution. In John R. Koza, editor, *Late Breaking Papers at the Genetic Programming 1997 Conference*, pages 280–286, Stanford University, California, July 1997. Stanford Bookstore.

[6] George Bilchev and Ian C. Parmee. The Ant Colony Metaphor for Searching Continuous Design Spaces. In Terence C. Fogarty, editor, *Evolutionary Computing*, pages 25–39. Springer Verlag, Sheffield, UK, April 1995.

[7] George Bilchev and Ian C. Parmee. Constrained and Multi-Modal Optimisation with an Ant Colony Search Model. In Ian C. Parmee and M. J. Denham, editors, *Proceedings of 2nd International Conference*

on Adaptive Computing in Engineering Design and Control. University of Plymouth, Plymouth, UK, March 1996.

[8] H. J. Bremermann. Optimization through evolution and recombination. In Marshall C. Yovitis and George T. Jacobi, editors, *Self-Organizing Systems*, pages 93–106. Spartan, Washington, D.C., 1962.

[9] Eduardo Camponogara and Sarosh N. Talukdar. A Genetic Algorithm for Constrained and Multiobjective Optimization. In Jarmo T. Alander, editor, *3rd Nordic Workshop on Genetic Algorithms and Their Applications (3NWGA)*, pages 49–62, Vaasa, Finland, August 1997. University of Vaasa.

[10] C. S. Chang, W. Wang, A. C. Liew, F. S. Wen, and D. Srinivasan. Genetic Algorithm Based Bicriterion Optimization for Traction Sustations in DC Railway System. In *Proceedings of the Second IEEE International Conference on Evolutionary Computation*, pages 11–16, Piscataway, New Jersey, 1995. IEEE Press.

[11] A. Charnes and W. W. Cooper. *Management Models and Industrial Applications of Linear Programming*, volume 1. John Wiley, New York, 1961.

[12] Chan-Jin Chung. *Knowledge-Based Approaches to Self-Adaptation in Cultural Algorithms*. PhD thesis, Wayne State University, Detroit, Michigan, 1997.

[13] Chan-Jin Chung and Robert G. Reynolds. A Testbed for Solving Optimization Problems Using Cultural Algorithms. In Lawrence J. Fogel, Peter J. Angeline, and Thomas Bäck, editors, *Evolutionary Programming V: Proceedings of the Fifth Annual Conference on Evolutionary Programming*, pages 225–236, Cambridge, Massachusetts, March 1996. MIT Press.

[14] Chan-Jin Chung and Robert G. Reynolds. CAEP: An Evolution-based Tool for Real-Valued Function Optimization using Cultural Algorithms. *Journal on Artificial Intelligence Tools*, 7(3):239–292, 1998.

[15] Jeffery K. Cochran, Shwu-Min Horng, and John W. Fowler. A Multi-Population Genetic Algorithm to Solve Multi-Objective Scheduling

Problems for Parallel Machines. Working Paper, 2000. (Submitted to *Computers and Operations Research*).

[16] Carlos A. Coello Coello and Nareli Cruz Cortés. A Parallel Implementation of an Artificial Immune System to Handle Constraints in Genetic Algorithms: Preliminary Results. In *Proceedings of the Congress on Evolutionary Computation 2002 (CEC'2002)*, volume 1, pages 819–824, Piscataway, New Jersey, May 2002. IEEE Service Center.

[17] Carlos A. Coello Coello and Efrén Mezura-Montes. Handling Constraints in Genetic Algorithms Using Dominance-Based Tournaments. In I.C. Parmee, editor, *Proceedings of the Fifth International Conference on Adaptive Computing Design and Manufacture (ACDM 2002)*, volume 5, pages 273–284, University of Exeter, Devon, UK, April 2002. Springer-Verlag.

[18] Carlos A. Coello Coello. Constraint-handling using an evolutionary multiobjective optimization technique. *Civil Engineering and Environmental Systems*, 17:319–346, 2000.

[19] Carlos A. Coello Coello. Treating Constraints as Objectives for Single-Objective Evolutionary Optimization. *Engineering Optimization*, 32(3):275–308, 2000.

[20] Carlos A. Coello Coello. Use of a Self-Adaptive Penalty Approach for Engineering Optimization Problems. *Computers in Industry*, 41(2):113–127, January 2000.

[21] Carlos A. Coello Coello. Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A survey of the state of the art. *Computer Methods in Applied Mechanics and Engineering*, 191(11-12):1245–1287, January 2002.

[22] Carlos A. Coello Coello and Gregorio Toscano Pulido. A Micro-Genetic Algorithm for Multiobjective Optimization. In Eckart Zitzler, Kalyanmoy Deb, Lothar Thiele, Carlos A. Coello Coello, and David Corne, editors, *First International Conference on Evolutionary Multi-Criterion Optimization*, pages 126–140. Springer-Verlag. Lecture Notes in Computer Science No. 1993, 2001.

[23] Carlos A. Coello Coello and Gregorio Toscano Pulido. Multiobjective Optimization using a Micro-Genetic Algorithm. In Lee Spector, Erik D. Goodman, Annie Wu, W.B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2001)*, pages 274–282, San Francisco, California, 2001. Morgan Kaufmann Publishers.

[24] Carlos A. Coello Coello, David A. Van Veldhuizen, and Gary B. Lamont. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Kluwer Academic Publishers, New York, May 2002. ISBN 0-3064-6762-3.

[25] Carlos Artemio Coello Coello. *An Empirical Study of Evolutionary Techniques for Multiobjective Optimization in Engineering Design*. PhD thesis, Department of Computer Science, Tulane University, New Orleans, LA, April 1996.

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

[27] David W. Corne, Nick R. Jerram, Joshua D. Knowles, and Martin J. Oates. PESA-II: Region-based Selection in Evolutionary Multiobjective Optimization. In Lee Spector, Erik D. Goodman, Annie Wu, W.B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2001)*, pages 283–290, San Francisco, California, 2001. Morgan Kaufmann Publishers.

[28] David W. Corne, Joshua D. Knowles, and Martin J. Oates. The Pareto Envelope-based Selection Algorithm for Multiobjective Optimization. In Marc Schoenauer, Kalyanmoy Deb, Günter Rudolph, Xin Yao, Evelyne Lutton, J. J. Merelo, and Hans-Paul Schwefel, editors, *Proceedings of the Parallel Problem Solving from Nature VI Conference*, pages 839–848, Paris, France, 2000. Springer. Lecture Notes in Computer Science No. 1917.

[29] I. Das and J. Dennis. A Closer Look at Drawbacks of Minimizing Weighted Sums of Objectives for Pareto Set Generation in Multicriteria Optimization Problems. *Structural Optimization*, 14(1):63–69, 1997.

[30] Lawrence Davis, editor. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, New York, 1991.

[31] Kalyanmoy Deb. *Binary and Floating-Point Function Optimzation using Messy Genetic Algorithms*. PhD thesis, University of Alabama, Tuscaloosa, AL 35487, 1991. Department of Engineering Mechanics.

[32] Kalyanmoy Deb. Multi-Objective Genetic Algorithms: Problem Difficulties and Construction of Test Problems. Technical Report CI-49/98, Dortmund: Department of Computer Science/LS11, University of Dortmund, Germany, 1998.

[33] Kalyanmoy Deb. Evolutionary Algorithms for Multi-Criterion Optimization in Engineering Design. In Kaisa Miettinen, Marko M. Mäkelä, Pekka Neittaanmäki, and Jacques Periaux, editors, *Evolutionary Algorithms in Engineering and Computer Science*, chapter 8, pages 135–161. John Wiley & Sons, Ltd, Chichester, UK, 1999.

[34] Kalyanmoy Deb. Multi-Objective Genetic Algorithms: Problem Difficulties and Construction of Test Problems. *Evolutionary Computation*, 7(3):205–230, Fall 1999.

[35] Kalyanmoy Deb. An Efficient Constraint Handling Method for Genetic Algorithms. *Computer Methods in Applied Mechanics and Engineering*, 186(2/4):311–338, 2000.

[36] Kalyanmoy Deb and David E. Goldberg. An investigation of niche and species formation in genetic function optimization. In James D. Schaffer, editor, *Proc. of the Third Int. Conf. on Genetic Algorithms*, pages 42–50, San Mateo, CA, 1989. Morgan Kaufmann.

[37] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA–II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, April 2002.

[38] Marco Dorigo and Gianni Di Caro. Ant colony optimization: A new meta-heuristic. In *1999 Congress on Evolutionary Computation*, pages 1470–1477, Piscataway, NJ, 1999. IEEE Service Center.

[39] Marco Dorigo and Gianni Di Caro. The ant colony optimization meta-heuristic. In David Corne, Marco Dorigo, and Fred Glover, editors, *New Ideas in Optimization*, pages 11–32. McGraw-Hill, London, 1999.

[40] Marco Dorigo and Luca Maria Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Trans. on Evolutionary Computation*, 1(1):53–66, 1997.

[41] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. Ant system: Optimization by a colony of cooperating agents. *IEEE Trans. on Systems, Man, and Cybernetics–Part B*, 26(1):29–41, 1996.

[42] W. H. Durham. *Co-evolution: Genes, Culture, and Human Diversity*. Stanford University Press, Stanford, California, 1994.

[43] Mark Erickson, Alex Mayer, and Jeffrey Horn. The Niched Pareto Genetic Algorithm 2 Applied to the Design of Groundwater Remediation Systems. In Eckart Zitzler, Kalyanmoy Deb, Lothar Thiele, Carlos A. Coello Coello, and David Corne, editors, *First International Conference on Evolutionary Multi-Criterion Optimization*, pages 681–695. Springer-Verlag. Lecture Notes in Computer Science No. 1993, 2001.

[44] Larry J. Eshelman and J. Davis Schaffer. Real-coded Genetic Algorithms and Interval-Schemata. In L. Darrell Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 187–202. Morgan Kaufmann Publishers, San Mateo, California, 1993.

[45] David B. Fogel. An analysis of evolutionary programming. In David B. Fogel and Wirt Atmar, editors, *Proc. of the First Annual Conference on Evolutionary Programming*, pages 43–51, La Jolla, CA, 1992. Evolutionary Programming Society.

[46] Lawrence J. Fogel, editor. *Evolutionary Computation. The Fossil Record. Selected Readings on the History of Evolutionary Algorithms*. The Institute of Electrical and Electronic Engineers, New York, 1998.

[47] Lawrence J. Fogel, Alvin J. Owens, and Michael J. Walsh. *Artificial Intelligence Through Simulated Evolution*. John Wiley and Sons, New York, 1966.

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

[49] Carlos M. Fonseca and Peter J. Fleming. Multiobjective Genetic Algorithms Made Easy: Selection, Sharing, and Mating Restriction. In *Proceedings of the First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*, pages 42–52, Sheffield, UK, September 1995. IEE.

[50] M. P. Fourman. Compaction of Symbolic Layout using Genetic Algorithms. In *Genetic Algorithms and their Applications: Proceedings of the First International Conference on Genetic Algorithms*, pages 141–153. Lawrence Erlbaum, 1985.

[51] Benjamin Franklin and Marcel Bergerman. Cultural algorithms: Concepts and experiments. In *Proc. of the 2000 Congress on Evolutionary Computation*, pages 1245–1251, Piscataway, NJ, 2000. IEEE Service Center.

[52] Xavier Gandibleux, Hiroyuki Morita, and Naoki Katoh. The Supported Solutions Used as a Genetic Information in a Population Heuristic. In Eckart Zitzler, Kalyanmoy Deb, Lothar Thiele, Carlos A. Coello Coello, and David Corne, editors, *First International Conference on Evolutionary Multi-Criterion Optimization*, pages 429–442. Springer-Verlag. Lecture Notes in Computer Science No. 1993, 2001.

[53] Fred Glover and Manuel Laguna. *Tabu Search*. Kluwer Academic Publishers, New York, 1998.

[54] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing Co., Reading, Massachusetts, 1989.

[55] David E. Goldberg, Kalyanmoy Deb, Hillol Kargupta, and Georges Harik. Rapid, accurate optimization of difficult problems using fast messy genetic algorithms. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 56–64, San Mateo, CA, 1993. Morgan Kaufman.

[56] Atidel Ben Hadj-Alouane and James C. Bean. A Genetic Algorithm for the Multiple-Choice Integer Program. *Operations Research*, 45:92–101, 1997.

[57] P. Hajela and J. Lee. Constrained Genetic Search via Schema Adaptation. An Immune Network Solution. In Niels Olhoff and George I. N. Rozvany, editors, *Proceedings of the First World Congress of Stuctural and Multidisciplinary Optimization*, pages 915–920, Goslar, Germany, 1995. Pergamon.

[58] P. Hajela and J. Lee. Constrained Genetic Search via Schema Adaptation. An Immune Network Solution. *Structural Optimization*, 12:11–15, 1996.

[59] P. Hajela and C. Y. Lin. Genetic search strategies in multicriterion optimal design. *Structural Optimization*, 4:99–107, 1992.

[60] P. Hajela and J. Yoo. Constraint Handling in Genetic Search Using Expression Strategies. *AIAA Journal*, 34(12):2414–2420, 1996.

[61] Robert Hinterding and Zbigniew Michalewicz. Your Brains and My Beauty: Parent Matching for Constrained Optimisation. In *Proceedings of the 5th International Conference on Evolutionary Computation*, pages 810–815, Anchorage, Alaska, May 1998.

[62] John H. Holland. Concerning efficient adaptive systems. In M. C. Yovitis, G. T. Jacobi, and G. D. Goldstein, editors, *Self-Organizing Systems*, pages 215–230. Spartan Books, Washington, D.C., 1962.

[63] John H. Holland. *Adaptation in Natural and Artifical Systems*. Ann Harbor, University of Michigan Press, 1975.

[64] Jeffrey Horn and Nicholas Nafpliotis. Multiobjective Optimization using the Niched Pareto Genetic Algorithm. Technical Report IlliGAl Report 93005, University of Illinois at Urbana-Champaign, Urbana, Illinois, USA, 1993.

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

[66] Y. Ijiri. *Management of Goals and Accounting for Control*. North-Holland, Amsterdan, 1965.

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

[68] W. Jakob, M. Gorges-Schleuter, and C. Blume. Application of Genetic Algorithms to task planning and learning. In R. Männer and B. Manderick, editors, *Parallel Problem Solving from Nature, 2nd Workshop*, Lecture Notes in Computer Science, pages 291–300, Amsterdam, 1992. North-Holland Publishing Company.

[69] Andrzej Jaszkiewicz. Genetic local search for multiple objective combinatorial optimization. Technical Report RA-014/98, Institute of Computing Science, Poznan University of Technology, 1998.

[70] Fernando Jiménez and José L. Verdegay. Evolutionary techniques for constrained optimization problems. In Hans-Jürgen Zimmermann, editor, *7th European Congress on Intelligent Techniques and Soft Computing (EUFIT'99)*, Aachen, Germany, 1999. Verlag Mainz. ISBN 3-89653-808-X.

[71] Xidong Jin and Robert G. Reynolds. Using Knowledge-Based Evolutionary Computation to Solve Nonlinear Constraint Optimization Problems: a Cultural Algorithm Approach. In *1999 Congress on Evolutionary Computation*, pages 1672–1678, Washington, D.C., July 1999. IEEE Service Center.

[72] Xidong Jin and Robert G. Reynolds. Mining Knowledge in Large-Scale Databases Using Cultural Algorithms with Constraint Handling Mechanisms. In *Proceedings of the Congress on Evolutionary Computation 2000 (CEC'2000)*, volume 2, pages 1498–1506, Piscataway, New Jersey, July 2000. IEEE Service Center.

[73] H. Jutler. Liniejnaja modiel z nieskolkimi celevymi funkcjami (liner model with several objective functions). *Ekonomika i matematiceckije Metody*, 3:397–406, 1967.

[74] S. Kazarlis and V. Petridis. Varying Fitness Functions in Genetic Algorithms: Studying the Rate of Increase of the Dynamic Penalty Terms. In A. E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, editors, *Proceedings of the 5th Parallel Problem Solving from Nature (PPSN V)*, pages 211–220, Heidelberg, Germany, September 1998. Amsterdan, The Netherlands, Springer-Verlag. Lecture Notes in Computer Science Vol. 1498.

[75] J.-H. Kim and H. Myung. Evolutionary programming techniques for constrained optimization problems. *IEEE Transactions on Evolutionary Computation*, 1:129–140, July 1997.

[76] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[77] Hajime Kita, Yasuyuki Yabumoto, Naoki Mori, and Yoshikazu Nishikawa. Multi-Objective Optimization by Means of the Thermodynamical Genetic Algorithm. In Hans-Michael Voigt, Werner Ebeling, Ingo Rechenberg, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature—PPSN IV*, Lecture Notes in Computer Science, pages 504–512, Berlin, Germany, September 1996. Springer-Verlag.

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

[79] Ryszard Kowalczyk. Constraint Consistent Genetic Algorithms. In *Proceedings of the 1997 IEEE Conference on Evolutionary Computation*, pages 343–348, Indianapolis, USA, April 1997. IEEE.

[80] Slawomir Koziel and Zbigniew Michalewicz. A Decoder-based Evolutionary Algorithm for Constrained Parameter Optimization Problems. In T. Bäck, A. E. Eiben, M. Schoenauer, and H.-P. Schwefel, editors, *Proceedings of the 5th Parallel Problem Solving from Nature (PPSN V)*, pages 231–240, Heidelberg, Germany, September 1998.

Amsterdan, The Netherlands, Springer-Verlag. Lecture Notes in Computer Science Vol. 1498.

[81] Slawomir Koziel and Zbigniew Michalewicz. Evolutionary Algorithms, Homomorphous Mappings, and Constrained Parameter Optimization. *Evolutionary Computation*, 7(1):19–44, 1999.

[82] Frank Kursawe. A variant of evolution strategies for vector optimization. In H. P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature. 1st Workshop, PPSN I*, volume 496 of *Lecture Notes in Computer Science*, pages 193–197, Berlin, Germany, oct 1991. Springer-Verlag.

[83] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Norwell, Massachusetts, 1993.

[84] T. Van Le. A Fuzzy Evolutionary Approach to Constrained Optimization Problems. In *Proceedings of the Second IEEE Conference on Evolutionary Computation*, pages 274–278, Perth, November 1995. IEEE.

[85] G. E. Liepins and Michael D. Vose. Representational Issues in Genetic Optimization. *Journal of Experimental and Theoretical Computer Science*, 2(2):4–30, 1990.

[86] Gunar E. Liepins and W. D. Potter. A Genetic Algorithm Approach to Multiple-Fault Diagnosis. In Lawrence Davis, editor, *Handbook of Genetic Algorithms*, chapter 17, pages 237–250. Van Nostrand Reinhold, New York, New York, 1991.

[87] Xiaojian Liu, D. W. Begg, and R. J. Fishwick. Genetic approach to optimal topology/controller design of adaptive structures. *International Journal for Numerical Methods in Engineering*, 41:815–830, 1998.

[88] Daniel H. Loughlin and S. Ranjithan. The Neighborhood constraint method: A Genetic Algorithm-Based Multiobjective Optimization Technique. In Thomas Bäck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 666–673, San Mateo, California, July 1997. Michigan State University, Morgan Kaufmann Publishers.

[89] Sushil Louis and Gregory Rawlins. Designer genetic algorithms: Genetic algorithms in structure design. In Richard K. Belew and Lashon B. Booker, editors, *Fourth International Conference on Genetic Algorithms*, pages 53–60, University of California, San Diego, jul 1991. Morgan Kauffman Publishers.

[90] Sushil J. Louis. *Genetic Algorithms as a Computational Tool for Design*. PhD thesis, Department of Computer Science, Indiana University, aug 1993.

[91] Sushil J. Louis and Fang Zhao. Domain knowledge for genetic algorithms. *International Journal of Expert Systems*, 8(3):195–211, 1995.

[92] Zbigniew Michalewicz. Genetic Algorithms, Numerical Optimization, and Constraints. In Larry J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA-95)*, pages 151–158, San Mateo, California, July 1995. University of Pittsburgh, Morgan Kaufmann Publishers.

[93] Zbigniew Michalewicz and Naguib F. Attia. Evolutionary Optimization of Constrained Problems. In *Proceedings of the 3rd Annual Conference on Evolutionary Programming*, pages 98–108. World Scientific, 1994.

[94] Zbigniew Michalewicz and Cezary Z. Janikow. Handling Constraints in Genetic Algorithms. In R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms (ICGA-91)*, pages 151–157, San Mateo, California, 1991. University of California, San Diego, Morgan Kaufmann Publishers.

[95] Zbigniew Michalewicz and G. Nazhiyath. Genocop III: A coevolutionary algorithm for numerical optimization with nonlinear constraints. In David B. Fogel, editor, *Proceedings of the Second IEEE International Conference on Evolutionary Computation*, pages 647–651, Piscataway, New Jersey, 1995. IEEE Press.

[96] Zbigniew Michalewicz and Marc Schoenauer. Evolutionary Algorithms for Constrained Parameter Optimization Problems. *Evolutionary Computation*, 4(1):1–32, 1996.

[97] Eric Michielssen, Jean-Michel Sajer, S. Ranjithan, and Raj Mittra. Design of Lightweight, Broad-Band Microwave Absorbers Using Genetic Algorithms. *IEEE Transactions on Microwave Theory and Techniques*, 41(6/7):1024–1031, 1993.

[98] Tom Mitchell. *Version Spaces: An Approach to Concept Learning*. PhD thesis, Computer Science Department, Stanford University, Stanford, California, 1978.

[99] Hyun Myung and Jong-Hwan Kim. Hybrid Interior-Lagrangian Penalty Based Evolutionary Optimization. In V.W. Porto, N. Saravanan, D. Waagen, and A.E. Eiben, editors, *Proceedings of the 7th International Conference on Evolutionary Programming (EP98)*, pages 85–94, Heidelberg, Germany, March 1998. San Diego, California, USA, Springer-Verlag. Lecture Notes in Computer Science Vol. 1447.

[100] C. K. Oei, D. E. Goldberg, and S.-J. Chang. Tournament Selection, Niching, and the Preservation of Diversity. Technical Report Technical Report 91011, University of Illinois at Urbana-Champaign, Urbana, Illinois, USA, 1993.

[101] A. Osyczka. *Multicriterion Optimization in Engineering with FORTRAN programs*. Ellis Horwood Limited, 1984.

[102] Andrzej Osyczka and Sourav Kundu. A new method to solve generalized multicriteria optimization problems using the simple genetic algorithm. *Structural Optimization*, 10:94–99, 1995.

[103] J. Paredis. Co-evolutionary Constraint Satisfaction. In *Proceedings of the 3rd Conference on Parallel Problem Solving from Nature*, pages 46–55, New York, 1994. Springer Verlag.

[104] V. Pareto. *Cours D'Economie Politique*, volume I and II. F. Rouge, Lausanne, 1975.

[105] I. C. Parmee and G. Purchase. The development of a directed genetic search technique for heavily constrained design spaces. In I. C. Parmee, editor, *Adaptive Computing in Engineering Design and Control-'94*, pages 97–102, Plymouth, UK, 1994. University of Plymouth.

[106] Ian C. Parmee and Andrew H. Watson. Preliminary Airframe Design Using Co-Evolutionary Multiobjective Genetic Algorithms. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'99)*, volume 2, pages 1657–1665, San Francisco, California, July 1999. Morgan Kaufmann.

[107] Carlo Poloni, Giovanni Mosetti, and Stefano Contessi. Multiobjective Optimization by GAs: Application to System and Component Design. In *Computational Methods in Applied Sciences '96: Invited Lectures and Special Technological Sessions of the Third ECCOMAS Computational Fluid Dynamics Conference and the Second ECCOMAS Conference on Numerical Methods in Engineering*, pages 258–264, Chichester, 1996. Wiley.

[108] David Powell, Michael Skolnick, and S. Tong. EnGENEous : domain independent, machine learning for design implementation. In J. David Schaffer, editor, *Third International Conference on Genetic Algorithms*, pages 151–9, George Mason University, jun 1989. Morgan Kauffman Publishers.

[109] David Powell and Michael M. Skolnick. Using genetic algorithms in engineering design optimization with non-linear constraints. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 424–431. Morgan Kaufmann Publishers, jul 1993.

[110] David J. Powell, Michael M. Skolnick, and Siu Shing Tong. Interdigitation : Hybrid technique for engineering design optimization employing genetic algorithms, expert systems, and numerical optimization. In Lawrence Davis, editor, *Handbook of Genetic Algorithms*, chapter 20, pages 312–331. Van Nostrand Reinhold, New York, 1991.

[111] Singiresu S. Rao. *Engineering Optimization*. John Wiley and Sons, third edition, 1996.

[112] Gregory J. Rawlins. Introduction. In Gregory J. Rawlins, editor, *Foundations of genetic algorithms*, pages 1–10. Morgan Kaufmann, San Mateo, CA, 1991.

[113] Tapabrata Ray, Tai Kang, and Seow Kian Chye. An Evolutionary Algorithm for Constrained Optimization. In Darrell Whitley, David Goldberg, Erick Cantú-Paz, Lee Spector, Ian Parmee, and Hans-Georg Beyer, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2000)*, pages 771–777, San Francisco, California, July 2000. Morgan Kaufmann.

[114] Ingo Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzbog, Stuttgart, 1973. German.

[115] Colin B. Reeves, editor. *Modern Heuristic Techniques for Combinatorial Problems*. John Wiley & Sons, Great Britain, 1993.

[116] A. C. Renfrew. Dynamic Modeling in Archaeology: What, When, and Where? In S. E. van der Leeuw, editor, *Dynamical Modeling and the Study of Change in Archaelogy*. Edinburgh University Press, Edinburgh, Scotland, 1994.

[117] Robert G. Reynolds. An Introduction to Cultural Algorithms. In A. V. Sebald and L. J. Fogel, editors, *Proceedings of the Third Annual Conference on Evolutionary Programming*, pages 131–139. World Scientific, River Edge, New Jersey, 1994.

[118] Robert G. Reynolds. Cultural algorithms: Theory and applications. In David Corne, Marco Dorigo, and Fred Glover, editors, *New Ideas in Optimization*, pages 367–377. McGraw-Hill, London, 1999.

[119] Robert G. Reynolds, Zbigniew Michalewicz, and M. Cavaretta. Using cultural algorithms for constraint handling in GENOCOP. In J. R. McDonnell, R. G. Reynolds, and D. B. Fogel, editors, *Proceedings of the Fourth Annual Conference on Evolutionary Programming*, pages 298–305. MIT Press, Cambridge, Massachusetts, 1995.

[120] Jon T. Richardson, Mark R. Palmer, Gunar Liepins, and Mike Hilliard. Some Guidelines for Genetic Algorithms with Penalty Functions. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms (ICGA-89)*, pages 191–197, San Mateo, California, June 1989. George Mason University, Morgan Kaufmann Publishers.

[121] Rodolphe G. Le Riche, Catherine Knopf-Lenoir, and Raphael T. Haftka. A Segregated Genetic Algorithm for Constrained Structural Optimization. In Larry J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA-95)*, pages 558–565, San Mateo, California, July 1995. University of Pittsburgh, Morgan Kaufmann Publishers.

[122] Jon Rowe, Kevin Vinsen, and Nick Marvin. Parallel GAs for Multiobjective Functions. In Jarmo T. Alander, editor, *Proceedings of the Second Nordic Workshop on Genetic Algorithms and Their Applications (2NWGA)*, pages 61–70, Vaasa, Finland, August 1996. University of Vaasa.

[123] Thomas P. Runarsson and Xin Yao. Stochastic Ranking for Constrained Evolutionary Optimization. *IEEE Transactions on Evolutionary Computation*, 4(3):284–294, September 2000.

[124] Eric Sandgren. Multicriteria design optimization by goal programming. In Hojjat Adeli, editor, *Advances in Design Optimization*, chapter 23, pages 225–265. Chapman & Hall, London, 1994.

[125] J. David Schaffer. Multiple Objective Optimization with Vector Evaluated Genetic Algorithms. In *Genetic Algorithms and their Applications: Proceedings of the First International Conference on Genetic Algorithms*, pages 93–100. Lawrence Erlbaum, 1985.

[126] Marc Schoenauer and Zbigniew Michalewicz. Evolutionary Computation at the Edge of Feasibility. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, *Proceedings of the Fourth Conference on Parallel Problem Solving from Nature (PPSN IV)*, pages 245–254, Heidelberg, Germany, September 1996. Berlin, Germany, Springer-Verlag.

[127] Marc Schoenauer and Spyros Xanthakis. Constrained GA Optimization. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms (ICGA-93)*, pages 573–580, San Mateo, California, July 1993. University of Illinois at Urbana-Champaign, Morgan Kauffman Publishers.

[128] R. Solich. Zadanie programowania liniowego z wieloma funkcjami celu (linear programming problem with several objective functions). *Przeglad Statystyczny*, 16:24–30, 1969.

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

[130] Patrick D. Surry and Nicholas J. Radcliffe. The COMOGA Method: Constrained Optimisation by Multiobjective Genetic Algorithms. *Control and Cybernetics*, 26(3):391–412, 1997.

[131] Patrick D. Surry, Nicholas J. Radcliffe, and Ian D. Boyd. A Multi-Objective Approach to Constrained Optimisation of Gas Supply Networks : The COMOGA Method. In Terence C. Fogarty, editor, *Evolutionary Computing. AISB Workshop. Selected Papers*, Lecture Notes in Computer Science, pages 166–180. Springer-Verlag, Sheffield, U.K., 1995.

[132] Gilbert Syswerda. Schedule Optimization Using Genetic Algorithms. In Lawrence Davis, editor, *Handbook of Genetic Algorithms*, chapter 21, pages 332–349. Van Nostrand Reinhold, New York, New York, 1991.

[133] C. H. Tseng and T. W. Lu. Minimax multiobjective optimization in structural design. *Int. J. Numerical Methods in Engineering*, 30:1213–1228, 1990.

[134] Effie Tsoi, Kit Po Wong, and Chun Che Fung. Hybrid GA/SA Algorithms for Evaluating Trade-off Between Economic Cost and Environmental Impact in Generation Dispatch. In David B. Fogel, editor, *Proceedings of teh Second IEEE Conference on Evolutionary Computation (ICEC'95)*, pages 132–137, Piscataway, New Jersey, 1995. IEEE Press.

[135] David 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, May 1999.

[136] David A. Van Veldhuizen and Gary B. Lamont. Evolutionary Computation and Convergence to a Pareto Front. In John R. Koza, editor, *Late Breaking Papers at the Genetic Programming 1998 Conference*, pages 221–228, Stanford University, California, July 1998. Stanford University Bookstore.

[137] David A. Van Veldhuizen and Gary B. Lamont. Multiobjective Optimization with Messy Genetic Algorithms. In *Proceedings of the 2000 ACM Symposium on Applied Computing*, pages 470–476, Villa Olmo, Como, Italy, 2000. ACM.

[138] Rémy Viennet, Christian Fontiex, and Ivan Marc. Multicriteria Optimization Using a Genetic Algorithm for Determining a Pareto Set. *Journal of Systems Science*, 27(2):255–260, 1996.

[139] Stefan Voget and Michael Kolonko. Multidimensional Optimization with a Fuzzy Genetic Algorithm. *Journal of Heuristics*, 4(3):221–244, September 1998.

[140] P. B. Wienke, C. Lucasius, and G. Kateman. Multicriteria target optimization of analytical procedures using a genetic algorithm. *Analytica Chimica Acta*, 265(2):211–225, 1992.

[141] Xiaofeng Yang and Mitsuo Gen. Evolution program for bicriteria transportation problem. In M. Gen and T. Kobayashi, editors, *Proceedings of the 16th International Conference on Computers and Industrial Engineering*, pages 451–454, Ashikaga, Japan, 1994. Pergamon Press.

[142] Gengui Zhou and Mitsuo Gen. Genetic Algorithm Approach on Multi-Criteria Minimum Spanning Tree Problem. *European Journal of Operational Research*, 114(1), April 1999.

[143] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. In K. Giannakoglou, D. Tsahalis, J. Periaux, P. Papailou, and T. Fogarty, editors, *EUROGEN 2001. Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems*, Athens, Greece, September 2001.

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

[145] Jesse B. Zydallis, David A. Van Veldhuizen, and Gary B. Lamont. A Statistical Comparison of Multiobjective Evolutionary Algorithms Including the MOMGA–II. In Eckart Zitzler, Kalyanmoy Deb, Lothar Thiele, Carlos A. Coello Coello, and David Corne, editors, *First International Conference on Evolutionary Multi-Criterion Optimization*, pages 226–240. Springer-Verlag. Lecture Notes in Computer Science No. 1993, 2001.