Resolución de problemas multi-objetivo a través de optimización mediante cúmulos de partículas

Tesis para obtener el grado de Maestro en Inteligencia Artificial
por

Maximino Salazar Lechuga

Maestría en Inteligencia Artificial
Facultad de Física e Inteligencia Artificial
Universidad Veracruzana - LANIA

Asesor:
Dr. Carlos A. Coello Coello
CINVESTAV - IPN

Revisor:
Dr. Carlos E. Mariano Romero
Instituto Mexicano de Tecnología del Agua

Revisor:
Dr. Manuel Martínez Morales
Maestría en Inteligencia Artificial - Universidad Veracruzana

Febrero, 2002

Resumen

En este trabajo de tesis se hace una propuesta para extender la heurística llamada "particle swarm optimization" (PSO) y así resolver problemas de optimización multiobjetivo. El método propuesto usa el concepto de dominancia de Pareto para determinar la dirección de vuelo de una partícula y mantiene vectores no-dominados encontrados previamente en un repositorio global que después es utilizado por otras partículas para guiar su vuelo. Esta propuesta es validada usando diversas funciones de prueba estándares en la literatura especializada. Nuestros resultados indican que el método es altamente competitivo con las técnicas evolutivas de optimización multiobjetivo actuales.

Agradecimientos

Quiero agradecer a Carlos Coello, mi asesor, por su invaluable apoyo, interés y motivación para guiarme a lo largo de este trabajo de investigación, por la confianza y valiosos consejos en momentos de caos, que me han ayudado a encontrar el orden y a descubrir un mundo maravilloso.

A todos mis amigos, familiares y maestros que de alguna u otra forma me han influenciado.

A los mejores padres que Dios (cualquiera que sea su representación) pudo poner en mi camino y sin los que nada de esto sería posible, realmente gracias Irma y Max.

A mi hermosa familia, Alicia (la más linda abuelita que nadie pudo haber tenido), Martha (gracias por tu amor de madre), Ale, Mimis y Jaime (mis hermanos), Meni (nunca podre agradecerte lo mucho que me quisiste), Maximino (siempre estarás en mi memoria), Ana y Angelina (gracias por su cariño), Mary, Jaime, Carlos, Miguel, a todas mis primas, primos, tías, tíos.

Agradezco el apoyo proporcionado por CONACyT a través de una beca (Número de Becario 141613) para cursar la maestría en Inteligencia Artificial de la Universidad Veracruzana y el LANIA, así como el apoyo para la realización de esta tesis a través del proyecto titulado "Nuevos Paradigmas en Optimización Evolutiva Multiobjetivo" (Ref. 34201-A) cuyo responsable es el Dr. Carlos A. Coello Coello.

Índice General

Índice de Figuras

Índice de Tablas

Introducción

La "optimización mediante cúmulos de partículas" (*Particle Swarm Optimization* o PSO), es una técnica de optimización de funciones no lineales de reciente creación. Esta técnica ha cautivado a diversos investigadores de computación evolutiva debido a su bajo costo computacional, su fácil implementación y su excelente desempeño.

Sin embargo, siendo ésta una técnica relativamente nueva (data de mediados de los 1990s), poco trabajo de investigación se ha realizado alrededor de ella y existe un sin número de temas que todavía no han sido abordados debidamente en la literatura. Uno de estos temas es el manejo de restricciones, ya que la técnica está diseñada como un algoritmo de optimización para problemas sin restricciones. Otro tema importante que no se había abordado es el de la optimización multiobjetivo, ya que la técnica se concibió para problemas mono-objetivo. En esta tesis nos hemos enfocado a abordar estos dos problemas: manejo de restricciones en problemas mono-objetivo y solución de problemas multiobjetivo sin restricciones.

La tesis se encuentra dividida en cinco capítulos cuya organización es la siguiente: El capítulo 1 introduce conceptos básicos de la computación evolutiva y describe las ramas que la conforman. Adicionalmente, muestra también lo que se considera como un nuevo paradigma dentro de la computación evolutiva: la PSO. El capítulo 2 describe la PSO, la forma en que trabaja y propone un algoritmo para que la técnica pueda resolver problemas restringidos. El capítulo 3 muestra los resultados obtenidos de los experimentos realizados al hacer la implementación del algoritmo propuesto en el capítulo 2. El capítulo 4 contiene la propuesta para hacer que la PSO pueda resolver problemas multiobjetivo. El capítulo 5 muestra los resultados obtenidos al realizar la implementación del algoritmo propuesto en el capítulo anterior y compara estos resultados contra dos técnicas que representan el estado del arte en optimización evolutiva multiobjetivo. Finalmente se presentan las conclusiones de nuestro trabajo, así como las líneas que consideramos más promisorias de trabajo futuro.

Capítulo 1

Conceptos Básicos

1.1 Introducción

En este capítulo se dará una breve descripción de lo que es la computación evolutiva y los paradigmas que comprende. También se hablará de una técnica desarrollada recientemente llamada "optimización mediante cúmulos de partículas" (*"Particle Swarm Optimization"* - PSO), y de la relación que ésta tiene con las técnicas de la computación evolutiva.

1.2 Computación Evolutiva

La computación evolutiva (CE) tiene sus raíces en el principio de la evolución de las especies, y está inspirada en gran parte en principios biológicos, el mecanismo de selección natural y la genética. La idea fundamental de la selección natural es que los individuos que son más aptos sobreviven al paso del tiempo y éstos a su vez heredan a sus antecesores las características que les ayudaron a prevalecer. Usualmente los paradigmas que comprende este campo se utilizan como procedimientos de búsqueda y/u optimización.

En su mayoría, la forma en que trabajan estos paradigmas es utilizando una población de puntos. Estos puntos generalmente reciben el nombre de individuos, los cuales evolucionan a lo largo de varias generaciones. Dichos individuos en realidad son vectores del hiper-espacio que se está explorando y representan posibles soluciones dentro del espacio de búsqueda.

Proceduralmente, la mayor parte de los paradigmas de la CE siguen los siguientes pasos:

1. Inicializan una población aleatoriamente.

2. Evalúan a cada uno de los individuos, asignándoles una aptitud.

3. Seleccionan a los progenitores de la nueva población con base en las aptitudes de cada individuo.

4. Se aplican operaciones evolutivas a la población, como cruza y mutación.

5. Se regresa al paso 2 hasta que se cumpla una condición de detención[1].

El área de la CE se encuentra dividida principalmente en 4 ramas:

- Algoritmos genéticos
- Programación evolutiva
- Estrategias evolutivas
- Programación genética

Y aunque existen otras formas de dividir a la CE[2], en [25] se sugiere esta clasificación y es la que aquí se tomará como base para hacer un recuento.

1.2.1 Algoritmos Genéticos

Esta rama de la CE es una de las más representativas del área, y también una dentro de la cual se ha realizado más trabajo. Uno de los principales precursores en esta área fue John Holland, quien junto con sus alumnos en la Universidad de Michigan, hizo importantes contribuciones al campo a principios de la década de 1960. Los algoritmos genéticos son algoritmos de búsqueda, que en cierta medida reflejan los procesos de la evolución. Este paradigma opera evaluando a una población de individuos, los cuales normalmente se representan mediante cromosomas haploides y son codificados usando cadenas de bits (aunque también existen implementaciones con números reales, enteros, caracteres, etc.). Con base en la evaluación de sus aptitudes, lo que se hace es seleccionar a los mejores individuos/cromosomas de la población para generar una nueva población. Es decir, de esta manera se asigna mayor probabilidad de selección a los más aptos para producir la siguiente generación; a esta nueva población de individuos/cromosomas se le aplican operadores de cruza y mutación y de esta forma se continúan modificando las cadenas de bits que codifican a los individuos/cromosomas esperando que al finalizar el ciclo los individuos hayan evolucionado lo suficiente como para haber encontrando una solución al menos sub-óptima.

La forma en la que generalmente trabajan los algoritmos genéticos es la siguiente:

1. Inicializan una población aleatoriamente.

2. Evalúan a cada uno de los individuos, asignándoles una aptitud.

3. Seleccionan a los progenitores de la nueva población con base en las aptitudes de cada individuo.

[1] Generalmente la condición es que el ciclo del algoritmo cumpla con un cierto número de iteraciones.

[2] Algunos autores incluyen a la programación genética dentro de los algoritmos genéticos, y algunos otros como en [20] agregan una rama más llamada "Sistemas de Clasificadores".

4. Se genera una nueva población aplicando operadores genéticos de cruza y mutación.

5. Se regresa al paso 2 hasta que se cumpla una condición de detención[3].

En [19] puede encontrarse una explicación mucho más detallada y a profundidad sobre cómo trabajan los algoritmos genéticos y las distintas formas en que operan.

1.2.2 Programación Evolutiva

El precursor principal de esta área es Lawrence J. Fogel, ya que él junto con sus colaboradores desarrollaron en 1960 la denominada programación evolutiva [14].

Los algoritmos genéticos y la programación evolutiva trabajan de forma muy parecida. La principal diferencia entre estas técnicas es de que la primera utiliza el operador de cruza y la segunda únicamente utiliza el operador de mutación. Esto se debe a que esta técnica, filosóficamente, considera a cada punto de la población como la representación de una especie, y no de un individuo. Como diferentes especies no pueden cruzarse entre sí, sólo se permite la mutación.

El algoritmo general de la programación evolutiva es el siguiente:

1. Inicializan una población aleatoriamente.

2. Se muta a cada individuo "padre" miembro de la población.

3. Se evalúan a todos los individuos (padres e hijos).

4. Se seleccionan a los miembros que formarán la nueva población (generalmente un proceso de selección mediante torneo).

5. Se regresa al paso 2 hasta que se cumpla una condición de detención[4].

1.2.3 Estrategias Evolutivas

Desde mediados de la década de 1960, Ingo Rechenberg y Hans-Paul Schwefel experimentaron con una técnica que sería conocida más tarde como estrategias evolutivas. A inicios de la década de 1970 Rechenberg publicó un libro que es considerado el pilar de esta técnica [31]. Hasta nuestros días las estrategias evolutivas han sido un campo con una gran actividad de investigación, principalmente en Europa.

Esta técnica comparte muchas características con la programación evolutiva (a pesar de que el desarrollo de cada una fue independiente por más de 30 años), así como de los algoritmos genéticos. Aunque la técnica utiliza operadores tanto de mutación como de cruza (generalmente este último operador en el campo de

[3] Generalmente la condición es que el ciclo del algoritmo cumpla con un cierto número de iteraciones, las cuales reciben el nombre de generaciones.

[4] Generalmente la condición es que el ciclo del algoritmo cumpla con un cierto número de iteraciones, las cuales reciben el nombre de generaciones.

las estrategias evolutivas recibe el nombre de recombinación), estos operadores funcionan de una manera un tanto distinta que en otras técnicas.

El siguiente procedimiento es el que utilizan generalmente las estrategias evolutivas:

1. Inicializan una población aleatoriamente.

2. Llevan a cabo la operación de recombinación usando μ padres para generar λ hijos.

3. Se efectúa la operación de mutación sobre los hijos.

4. Se evalúan los hijos, o ambos padres e hijos (μ, λ, o $\mu + \lambda$).

5. Se seleccionan a los μ individuos que formarán la nueva población.

6. Se regresa al paso 2 hasta que se cumpla una condición de detención[5].

1.2.4 Programación Genética

La programación genética tuvo su origen a finales de la década de 1950, con algunos trabajos hechos por Friedberg [17], y sus colaboradores [18]; aunque desafortunadamente estos primeros esfuerzos no fueron del todo satisfactorios. Más tarde, Nichal Lynn Cramer [6] hizo algunos experimentos en los que tampoco obtuvo buenos resultados debido a que la función de aptitud que utilizaba en su propuesta no se encontraba automatizada. Fue hasta finales de la década de 1980 cuando un estudiante de Holland, John Koza desarrolló la programación genética en la forma en que es conocida en la actualidad [27].

Aunque algunos consideran a este paradigma una subrama de los algoritmos genéticos, éstos tienen diferencias significativas como las siguientes:

- La función principal de este paradigma es la de evolucionar programas computacionales, los cuales usualmente son representados como estructuras de árboles que pueden ser ejecutadas. En contraste, los algoritmos genéticos, como ya se mencionó, usan típicamente cadenas de bits que representan vectores.

- Las aptitudes de estas estructuras/individuos (programas computacionales) se evalúan ejecutando los programas que representan. Esto contrasta con los algoritmos genéticos en donde la aptitud se mide con base en la función que representa el problema.

En suma, el objetivo de la programación genética es el de encontrar un programa computacional que arroje los resultados deseados dado un conjunto de entradas, y esto como consecuencia limita a este paradigma al espacio de los programas.

El procedimiento que sigue la programación genética es el siguiente:

[5] Generalmente la condición es que el ciclo del algoritmo cumpla con un cierto número de iteraciones, las cuales reciben el nombre de generaciones.

1. Inicializan una población de árboles (programas computacionales).

2. Evalúan a cada uno de los árboles, asignándoles una aptitud.

3. Generan una nueva población de árboles con base en su aptitud mediante cruza (y en ocasiones se utiliza un segundo operador, como la mutación o permutación de ramas).

4. Se regresa al paso 2 hasta que se cumpla una condición de detención[6].

1.3 Optimización Mediante Cúmulos de Partículas

En 1995 Kennedy y Eberhart [24] propusieron una nueva técnica de optimización llamada "optimización mediante cúmulos de partículas" (*"Particle Swarm Optimization"* - PSO). La idea central de este paradigma es la de simular los movimientos de un grupo de aves que intentan encontrar comida. La técnica puede ser vista como un algoritmo de comportamiento distribuido que lleva a cabo una búsqueda multidimensional (en una versión generalizada).

La técnica de PSO, al igual que las técnicas que la computación evolutiva comprende, utiliza una población de posibles soluciones, las cuales irán evolucionando hasta encontrar una solución óptima (o lo más cercanamente posible) al problema presentado, y su grado de aptitud es medido por una función definida para el problema. Así como existen similitudes con los otros paradigmas también existen diferencias, como la de que los individuos en vez de recibir ese nombre dentro de este paradigma reciben el de partículas, las cuales sobre-vuelan las posibles soluciones a través del hiper-espacio (el espacio de búsqueda del problema). A estas partículas cuando se les inicializa, se les asigna además una velocidad de manera estocástica y de igual forma ésta irá variando a lo largo del ciclo de ejecución del programa. Esta velocidad es con la que las partículas serán aceleradas y mediante la cual serán capaces de explorar el hiper-espacio[7].

1.4 Computación Evolutiva y PSO (Conclusión)

Como ya se mencionó, los paradigmas que comprende la computación evolutiva y la técnica de PSO tienen algunas diferencias pero también comparten algunas similitudes que hacen o sugieren que la técnica PSO puede ser etiquetada como una técnica evolutiva (ver sección 1.3). De hecho, en ocasiones la técnica de PSO es considerada como el quinto paradigma de la computación evolutiva[25].

Para finalizar este capítulo valdría la pena citar a Kennedy y Eberhart:

[6] Generalmente la condición es que el ciclo del algoritmo cumpla con un cierto número de iteraciones, las cuales reciben el nombre de generaciones.

[7] En capítulos posteriores se hablará más a detalle sobre cómo opera este paradigma. Aquí sólo se hace una pequeña introducción del mismo para poder relacionarlo con las técnicas que abarca la computación evolutiva.

"La comunidad de CE ha dado la bienvenida a la técnica de PSO como una forma alterna de hacer las cosas, y los investigadores del PSO se consideran copartícipes en una importante empresa, al punto de considerar a la PSO como una especie de algoritmo evolutivo." [8]

[8] Tomado de [25, pp. 361], traducción libre.

Capítulo 2

Optimización mediante cúmulos de partículas

2.1 Introducción

En este capítulo nos adentraremos a la técnica llamada "optimización mediante cúmulos de partículas" (*"Particle Swarm Optimization"* - PSO). Hablaremos un poco acerca de sus orígenes, su forma de operación, su funcionamiento, y la forma en la que puede ser extendida para resolver problemas con restricciones.

2.2 Antecedentes

A continuación daremos algunos detalles sobre los orígenes de la PSO y las diferentes ramas de la ciencia que la han influenciado.

2.2.1 Orígenes

La PSO se refiere a un algoritmo de optimización numérica, y goza actualmente de un gran auge a pesar de su reciente creación. Esta técnica evolutiva fue desarrollada por James Kennedy y Russell Eberhart, y se dio a conocer en un artículo que publicaron en 1995 [24]. En ese artículo Kennedy y Eberhart hablan principalmente de la forma en que surgió el PSO, ya que este paradigma básicamente emerge del trabajo que se encontraban realizando con un simulador de conductas sociales que se llevan a cabo en agrupaciones de aves y/o peces principalmente. De este simulador es de donde surge la idea de volverlo un optimizador, debido a la forma en que vieron que éste se comportaba.

2.2.2 Influencias

Cabe mencionar que aunque el paradigma de la PSO es notablemente similar en algunos aspectos a los de la computación evolutiva (CE), y como ya se mencionó forma parte de éstos (ver sección 1.4), este paradigma se encuentra influenciado fuertemente por algunas otras corrientes de entre las que destacan: vida artificial, psicología social, ingeniería y ciencias de la computación. También se encuentra estrechamente relacionado con los autómatas celulares, ya que como lo mencionan Kennedy y Eberhart en [25], y de acuerdo al matemático Rudy Rucker en [32]:

> "... los autómatas celulares tienen tres atributos principales: (1) la actualización de las células se lleva a cabo en paralelo, (2) el nuevo valor de cada célula depende sólo del valor anterior de la célula y de sus vecinos, y (3) todas las células son actualizadas usando las mismas reglas ..." [1]

En este punto tal vez no sea del todo obvio, pero más adelante será claro el hecho de que a las partículas dentro de la PSO se les podría conceptualizar como células dentro de los autómatas celulares.

2.3 PSO

En esta sección veremos la forma en que el paradigma de la PSO funciona y además observaremos su desempeño en un problema sencillo de optimización para ver la forma en que es capaz de resolverlo.

2.3.1 Descripción del algoritmo

Como ya se ha discutido (ver sección 1.3), la PSO es similar a los paradigmas de la CE en que se inicializa una población de posibles soluciones aleatoriamente. Sin embargo, a diferencia de éstos, a las soluciones en la PSO (partículas) se les asigna una velocidad inicial (también de manera aleatoria). Además, cabe señalar que, cada partícula dentro de la PSO va manteniendo un registro de la posición en donde ha encontrado una mejor solución para el problema que se encuentre resolviendo. Así mismo, también se va teniendo un registro de la mejor solución global que las partículas han encontrado en su conjunto. Y es mediante esa velocidad, el registro de su mejor posición previa y el registro de la mejor posición global, que las partículas se acelerarán y volarán dentro del espacio de búsqueda del problema, explorando diferentes soluciones. Para ver más claramente la forma en que funciona la PSO veamos el proceso original, tal y como lo muestran Eberhart y Shi en [13], para implementar la PSO:

1. Inicializar una población de partículas con posiciones y velocidades aleatorias en las d dimensiones dentro del espacio del problema.

2. Evaluar el grado de aptitud con la función a optimizar y con las d variables de decisión, para cada partícula.

[1] Tomado de [25, pp. xvii], traducción libre.

3. Comparar las aptitudes de las partículas evaluadas con el mejor (*pbest*) de cada partícula. Si el valor actual es mejor que *pbest*, entonces actualizar el valor de *pbest* con el valor actual de la partícula, y la posición de *pbest* igualarla a la posición actual en el espacio d-dimensional.

4. Comparar las aptitudes con la mejor posición de toda la población encontrada anteriormente (*gbest*). Si el valor actual es mejor que el *gbest* registrado, entonces actualizar *gbest* con el actual valor y posición de la partícula.

5. Cambiar la velocidad y posición de la partícula de acuerdo a las ecuaciones (2.1) y (2.2), respectivamente:

$$\begin{aligned} VEL[i] &= VEL[i] + C_1 * R_1 * (PBESTS[i] - POP[i]) + \\ &\quad C_2 * R_2 * (PBESTS[GBEST] - POP[i]) \quad (2.1) \\ POP[i] &= POP[i] + VEL[i] \quad (2.2) \end{aligned}$$

6. Ir al paso 2 hasta que el criterio de detención se cumpla. Dicho criterio de detención es usualmente un valor suficientemente bueno para la aptitud o un máximo número de iteraciones (generaciones).

Este procedimiento es la forma más general en la que trabaja la PSO. Existen varias modificaciones a este procedimiento[2], pero en esencia el procedimiento anterior es la base de cualquier implementación de la técnica. De hecho, este procedimiento es la versión global de la PSO[3], debido a que todas las partículas en la fórmula de la velocidad (2.1) son atraídas o guiadas hacia la zona en que se encuentra el óptimo global *gbest*; mientras que existe una versión local, en la que las partículas además de guiarse por un mejor global se guían por las mejores posiciones encontradas por algún(os) vecino(s) que se encuentre(n) ligado(s) topológicamente a las partículas.

Algo sobre lo que también valdría la pena comentar un poco es sobre los coeficientes utilizados en la ecuación (2.1), para calcular la velocidad; los coeficientes R_1 y R_2, se introdujeron en el algoritmo por la necesidad de lograr que las partículas se desplacen una distancia determinada (aleatoria) hacia donde se creía que podían tener una mejora y es por eso que ambas variables toman un valor aleatorio entre 0 y 1; similarmente C_1 y C_2 es el grado de velocidad máxima hacia donde se moverán las partículas. Los valores asignados a las variables C_1 y C_2 usualmente son constantes a lo largo del ciclo de vida del algoritmo, más sin embargo éstos pueden variar en cada ejecución. Existe un artículo publicado por Shi y Eberhart [36] en donde sugieren algunos valores recomendables para estas variables.

[2] Es casi inevitable que al realizar alguien una implementación de un algoritmo éste plasme su toque personal.

[3] Queremos mencionar que a lo largo de la tesis se ha trabajado únicamente con la versión global de la PSO, por lo que siempre haremos referencia a esta versión, a menos que hagamos mención específica de alguna otra.

Tamaño del cúmulo	Número de iteraciones	W	$C1$	$C2$
120	200	0.4	1.4	1.4

Tabla 2.1: Parámetros utilizados en el experimento simple.

Mejor solución	0.998004
Peor solución	0.998004
Media	0.998004
Varianza	0.000000
Desv. Est.	0.000000
Mediana	0.998004

Tabla 2.2: Estadísticas de las 20 corridas del experimento simple.

2.3.2 Experimento simple

Ahora veremos el comportamiento de la técnica de PSO, descrita en la sección anterior, al realizar un ejercicio de optimización sobre la función F5 de De Jong [7]:

$$\frac{1}{1/K + \sum_{j=1}^{25} f_j^{-1}(x_1, x_2)} \tag{2.3}$$

donde:

$$f_j(x_1, x_2) = c_j + \sum_{i=1}^{2} (x_i - a_{ij})^6$$

donde:

$$-65.535 \leq x_i \leq 65.536, K = 500, c_j = j,$$

y:

$$[a_{ij}] = \begin{bmatrix} -32 & -16 & 0 & 16 & 32 & -32 & -16 & \ldots & 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & -16 & -16 & \ldots & 32 & 32 & 32 \end{bmatrix}$$

Esta función tiene un óptimo global de 0.998 en el punto $(x_1, x_2) = (-32, -32)$.

En la tabla 2.3 podemos observar los resultados arrojados por nuestra implementación a la técnica PSO. En dicha tabla se ven los mejores valores que se obtuvieron al final de cada una de las 20 corridas realizadas en el experimento. Los parámetros que se utilizaron para cada una de estas corridas se encuentran en la tabla 2.1. Cabe hacer mención de que estos parámetros fueron escogidos en base a un proceso de ajuste empírico, el cual nos llevo a escoger dichos valores.

En la tabla 2.2 se encuentran las estadísticas generadas con los resultados objeto de las corridas. Por último, la figura 2.1 muestra una gráfica de convergencia que corresponde a la corrida número 1 de esta serie de corridas.

Corrida	x1	x2	f(x)
1	-31.98	-31.98	0.998
2	-31.98	-31.98	0.998
3	-31.98	-31.98	0.998
4	-31.98	-31.98	0.998
5	-31.98	-31.98	0.998
6	-31.98	-31.98	0.998
7	-31.98	-31.98	0.998
8	-31.98	-31.98	0.998
9	-31.98	-31.98	0.998
10	-31.98	-31.98	0.998
11	-31.98	-31.98	0.998
12	-31.98	-31.98	0.998
13	-31.98	-31.98	0.998
14	-31.98	-31.98	0.998
15	-31.98	-31.98	0.998
16	-31.98	-31.98	0.998
17	-31.98	-31.98	0.998
18	-31.98	-31.98	0.998
19	-31.98	-31.98	0.998
20	-31.98	-31.98	0.998

Tabla 2.3: Soluciones encontradas para el experimento simple en 20 corridas.

2.4 PSO para problemas con restricciones

Pasaremos ahora a discutir la manera en que esta técnica puede ser extendida para resolver problemas con restricciones.

A continuación mostraremos el procedimiento que sugerimos para incorporar restricciones (claro que pueden existir otros tal vez más eficientes). Posteriormente profundizaremos en las partes del procedimiento que hayan cambiado de la versión original del PSO a ésta:

1. Inicializamos aleatoriamente una población POP de partículas d dimensionales junto con sus velocidades, donde MAX es el número de partículas del cúmulo:

   (a) Para $i = 0$, hasta MAX

   Inicializa $POP[i]$

   Inicializa $VEL[i]$

2. Evaluamos a cada una de las partículas en POP.

3. Obtenemos a la mejor partícula del cúmulo POP y se almacena su valor y posición en $GBEST$.

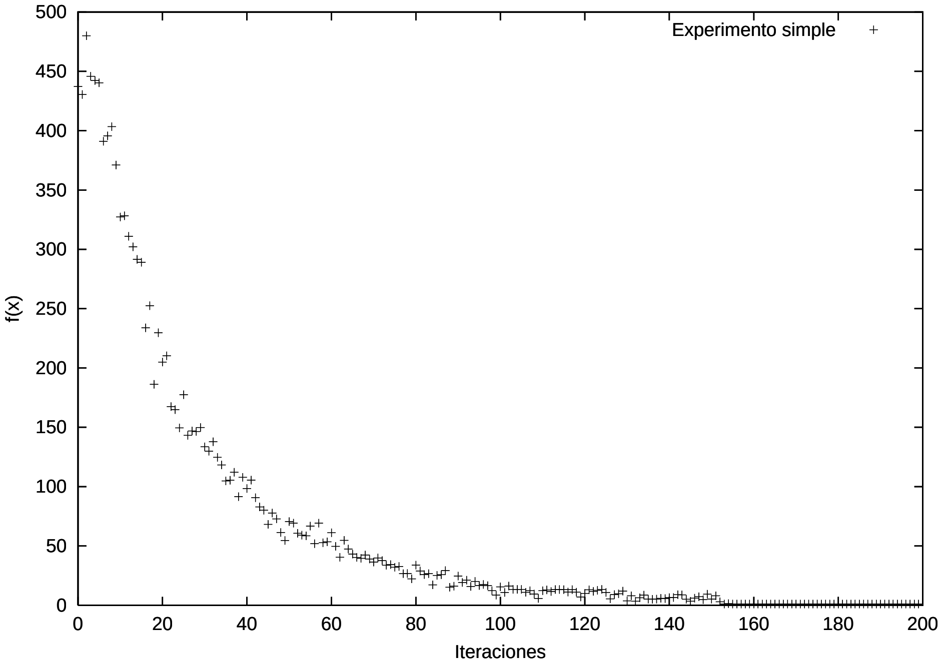

Figura 2.1: Gráfica de convergencia de la corrida 1 (Experimento simple).

4. Inicializamos la memoria de cada partícula con ella misma (esta memoria servirá de guía en la búsqueda, ya que ésta, junto con la mejor partícula global, serán utilizadas como referencias para viajar por el espacio de búsqueda).

   (a) Para $i = 0$, hasta MAX

   $PBESTS[i] = POP[i]$

5. Mientras no se cumpla con el número de ciclos especificado:

   (a) Calculamos la velocidad y posición de cada partícula con las ecuaciones (2.4) y (2.5) respectivamente:

$$\begin{aligned} VEL[i] &= VEL[i] + C_1 * R_1 * (PBESTS[i] - POP[i]) + \\ &\quad C_2 * R_2 * (PBESTS[GBEST] - POP[i]) \qquad (2.4) \\ POP[i] &= POP[i] + VEL[i] \qquad (2.5) \end{aligned}$$

   (b) Se mantienen las partículas dentro del espacio de búsqueda. Esto es, en caso de que las partículas hayan salido del hipercubo en el que se encuentran realizando la búsqueda, se reintegran de nuevo a él.

   (c) Se evalúan cada una de las partículas POP.

(d) Cuando la posición actual de la partícula sea mejor que la que contiene en su memoria, ésta se actualiza de la siguiente manera:

$$PBESTS[i] = POP[i]$$

En caso de que no sea mejor, continúa con la mejor posición encontrada con anterioridad.

(e) Buscamos la mejor posición encontrada hasta ahora, que será la que guiará la búsqueda en la siguiente iteración del ciclo. El líder $GBEST$ se busca en $PBESTS[i]$ que es donde se encuentran las mejores posiciones encontradas.

(f) Aumentamos el ciclo.

6. Fin del Mientras.

A simple vista pareciera que el procedimiento es muy similar al que presentamos en la sección 2.3.1, excepto por algunas sutiles diferencias que a continuación explicaremos.

Veamos la forma en que las partículas son evaluadas en el algoritmo propuesto en esta sección. Bajo este esquema, las partículas deben de competir de dos formas, una es contra sí mismas y la otra contra las demás.

Cuando se compara o compite contra sí misma una partícula lo hace de la siguiente manera: calculamos la magnitud de restricciones violadas de la partícula en la posición actual. En el algoritmo original cada partícula tiene una memoria que nos indica cuál ha sido su mejor posición durante el ciclo. Por lo tanto, también se calcula la magnitud de violación de restricciones en la posición almacenada; una vez que se tienen esos valores lo primero que se hace es ver si sus magnitudes son cero (ninguna posición violó restricciones). En caso de que así sea, la posición que se retendrá o se almacenará en la memoria de la partícula será la que cuente con una mejor aptitud; en caso de que esta situación no se presente, se verifica si alguna de las dos cuenta con una magnitud de violación igual a cero. En caso de ser así, se retiene a la que cuente con la magnitud de cero. Si esta situación tampoco se presenta, se verifica cuál de las dos es la que viola en menor magnitud las restricciones: ¿la mejor posición durante todo el ciclo de esa partícula o la posición actual de la partícula?. En caso de que la posición actual de la partícula tenga una menor magnitud de violación de restricciones, esa pasará a ser la nueva posición que deberá retener en memoria la partícula, no tomando en cuenta la aptitud que estas dos tengan. En el caso contrario, (o sea, cuando la posición actual de la partícula tenga una mayor magnitud de violación de restricciones que la posición memorizada) esta última no se verá afectada. En el caso de que ambas posiciones (la memorizada y la actual) tengan la misma magnitud de violación de restricciones lo que se hace para ver a cuál se retendrá en la memoria de la partícula es tomar a la partícula con el valor más alto de aptitud. Este tipo de competencia se da en el paso (5d).

El otro tipo de competencia (la competencia global), es la que se da para obtener al líder de la población (el que guiará la búsqueda). Aquí lo que se hace es obtener en cada ciclo a la mejor posición encontrada hasta ese momento. Dado que las mejores posiciones que han encontrado las partículas se encuentran en sus memorias, lo que hacemos es buscar al líder dentro de sus memorias. Para hacer esto, calculamos las magnitudes de violación de restricciones para cada posición guardada en la memoria de cada partícula. Una vez que se tienen esas magnitudes se toma como aspirante a líder a la posición memorizada con la menor magnitud de violación. En caso de que esa magnitud no sea igual a cero (si no es igual a cero quiere decir que viola restricciones, como se mencionó anteriormente) esa es la posición que guiará la búsqueda, debido a que de todas es la que más cercana podría encontrarse a la zona factible. En caso de que el aspirante a líder cuente con una magnitud de violación igual a cero, tomaremos como segundo aspirante a la siguiente posición memorizada con menor magnitud. Si también ese segundo aspirante tiene una magnitud de violación igual a cero, se tomará como aspirante ganador al que cuente con una mayor aptitud. Este último paso se repetirá hasta que se encuentra una magnitud que sea distinta a cero, o hasta que se hayan terminado de evaluar a todas las posiciones memorizadas por las partículas. Al finalizar, el líder será el que haya permanecido como aspirante ganador. Este tipo de competencia es la que se lleva a cabo en los pasos (3), y (5e).

Capítulo 3

Resolución de problemas con restricciones a través de PSO (experimentos)

3.1 Introducción

Dentro de este capítulo se mostrarán algunos resultados obtenidos al optimizar problemas que contienen restricciones mediante el algoritmo propuesto en el capítulo anterior (ver sección 2.4), y además se discutirán los resultados obtenidos.

3.2 Experimentos de problemas con restricciones

En el capítulo anterior propusimos un algoritmo que extendía la técnica PSO. Esta extensión se hizo con el fin de que la PSO fuera capaz de resolver problemas con restricciones (ver sección 2.4); a continuación, se presentarán algunos experimentos realizados para probar la eficiencia del algoritmo propuesto.

3.2.1 Experimento 1

En este primer experimento mostramos los resultados que arroja la implementación de nuestra propuesta para una función de prueba cuyo propósito es el diseño de un contenedor de presión. La función es la siguiente:

Minimizar :

$$F(X) = 0.6224x_1x_3x_4 + 1.7781x_2x_3^2 + 3.1661x_1^2x_4 + 19.84x_1^2x_3 \tag{3.1}$$

Variables de Diseño	**SA[39]**	**GeneAS[10]**	**Kannan[23]**	**Sandgren[34]**
	Mejor solución encontrada			
$x_1(T_s)$	1.125000	0.937500	1.125000	1.125000
$x_2(T_h)$	0.625000	0.500000	0.625000	0.625000
$x_3(R)$	58.290000	48.329000	58.291000	47.700000
$x_4(L)$	43.696000	112.679000	43.690000	117.701000
$g1(\vec{x})$	-0.000003	-0.004750	0.000016	-0.204390
$g2(\vec{x})$	-0.068913	-0.038941	-0.068904	-0.169942
$g3(\vec{x})$	-26.566438	-3652.876838	-21.220104	54.226012
$g4(\vec{x})$	-196.304000	-127.321000	-196.310000	-122.299000
$f(\vec{x})$	**7198.126500**	**6410.381100**	**7198.042800**	**8129.103600**

Tabla 3.1: Mejores Soluciones reportadas en la literatura para la función del experimento 1. **Parte I**

Variables de Diseño	**MVEP[1]**	**IDCNLPC[22]**	**Coello[5]**
	Mejor solución encontrada		
$x_1(T_s)$	1.000000	1.125000	0.812500
$x_2(T_h)$	0.625000	0.625000	0.437500
$x_3(R)$	51.195800	48.380700	40.323900
$x_4(L)$	90.782100	111.744900	200.000000
$g1(\vec{x})$	-0.011921	-0.191252	-0.034324
$g2(\vec{x})$	-0.136592	-0.163448	-0.052847
$g3(\vec{x})$	-13584.583968	-72.970137	-27.105845
$g4(\vec{x})$	-149.217900	-128.255100	-40.000000
$f(\vec{x})$	**7108.616000**	**8049.341100**	**6288.744500**

Tabla 3.2: Mejores Soluciones reportadas en la literatura para la función del experimento 1. **Parte II**

Sujeto a :

$$g1(X) = -x_1 + 0.0193x_3 \leq 0 \quad (3.2)$$
$$g2(X) = -x_2 + 0.00954x_3 \leq 0 \quad (3.3)$$
$$g3(X) = -\pi x_3^2 x_4 - \frac{4}{3}\pi x_3^3 + 1296000 \leq 0 \quad (3.4)$$
$$g4(X) = x_4 - 240 \leq 0 \quad (3.5)$$

donde :

$$1 \leq x_1, x_2 \leq 99 \text{ y } 10 \leq x_3, x_4 \leq 200$$

Esta función, así como los mejores resultados reportados en la literatura (tablas 3.1 y 3.2), se encuentran en [3].

Como podemos observar, la función cuenta con cuatro variables de decisión y cuatro restricciones.

Corrida	x1	x2	x3	x4	f(x)	Factible
1	13.23	15.71	41.06	189.95	7689.67	$\in$
2	20.38	11.00	61.88	25.21	7512.80	$\in$
3	15.93	8.25	50.84	91.81	6163.72	$\notin$
4	14.95	7.41	45.29	140.78	6096.41	$\in$
5	13.16	7.45	41.12	189.18	6183.41	$\in$
6	13.46	7.66	41.29	187.00	6162.66	$\in$
7	16.52	8.82	49.40	103.22	6649.56	$\in$
8	16.85	8.28	51.18	89.29	6471.59	$\in$
9	15.81	8.48	46.74	126.49	6559.60	$\in$
10	15.88	7.80	45.80	135.62	6431.80	$\in$
11	20.82	10.99	63.15	19.26	7430.72	$\in$
12	15.78	7.35	45.58	137.84	6460.98	$\in$
13	13.95	7.27	41.97	178.23	6075.74	$\in$
14	14.54	7.33	43.36	161.92	6336.39	$\in$
15	13.30	7.93	41.44	185.02	6142.05	$\in$
16	17.13	9.98	54.93	63.45	6780.81	$\in$
17	13.86	29.59	41.42	185.23	10339.02	$\in$
18	18.04	9.21	57.70	47.36	6882.37	$\in$
19	15.80	8.56	47.86	116.30	6442.48	$\in$
20	14.08	7.82	44.63	147.64	6173.46	$\in$

Tabla 3.3: Soluciones encontradas para el experimento 1 en 20 corridas.

Tamaño del cúmulo	Número de iteraciones	W	$C1$	$C2$
120	200	0.2	1.2	1.2

Tabla 3.4: Parámetros utilizados en el experimento 1.

En la tabla 3.3 se muestran los resultados de veinte corridas, ejecutadas con los parámetros que se encuentran en la tabla 3.4 para todas las corridas; es decir, en la tabla 3.3 se encuentran resumidos los mejores resultados obtenidos en cada una de las veinte corridas y además observamos los valores de cada una de las variables, para cada una de las corridas y el valor que la función toma. En esa misma tabla también encontramos un campo que nos indica si el resultado es una solución factible o no; esto último está representado mediante el símbolo de pertenencia.

Además, en la tabla 3.5, podemos observar los valores obtenidos para cada una de las restricciones impuestas por este problema de ingeniería. Y al igual que en la tabla 3.3, se muestra un campo en el que se indica si la solución generada en esa corrida fue factible o no.

Finalmente, en la tabla 3.6 se muestran algunos datos estadísticos generados con los resultados de las veinte corridas. Y en la figura 3.1 vemos la gráfica de convergencia correspondiente a la corrida número 12, que es la solución más cercana a la mediana.

Corrida	g1	g2	g3	g4	Factible
1	-0.02	-0.55	-129.16	-50.05	$\in$
2	-0.06	-0.03	-1.43	-214.79	$\in$
3	0.04	-0.01	-23.05	-148.19	$\notin$
4	-0.00	-0.01	-15.75	-99.22	$\in$
5	-0.02	-0.05	-246.78	-50.82	$\in$
6	-0.02	-0.04	-489.24	-53.00	$\in$
7	-0.05	-0.03	-95.44	-136.78	$\in$
8	-0.01	-0.01	-469.17	-150.71	$\in$
9	-0.04	-0.05	-61.92	-113.51	$\in$
10	-0.05	-0.00	-18.00	-104.38	$\in$
11	-0.03	-0.02	-104.21	-220.74	$\in$
12	-0.06	-0.00	-471.49	-102.16	$\in$
13	-0.00	-0.04	-152.14	-61.77	$\in$
14	-0.04	-0.02	-1529.24	-78.08	$\in$
15	-0.01	-0.04	-1.51	-54.98	$\in$
16	-0.00	-0.04	-43.85	-176.55	$\in$
17	-0.01	-1.42	-87.97	-54.77	$\in$
18	-0.01	-0.01	-4157.73	-192.64	$\in$
19	-0.01	-0.04	-108.55	-123.70	$\in$
20	-0.01	-0.01	-121.14	-92.36	$\in$

Tabla 3.5: Valores de las restricciones para el experimento 1 de las 20 corridas realizadas.

Mejor solución	6075.7409
Peor solución	6163.7171
Media	6749.2619
Varianza	947132.9152
Desv. Est.	973.2075
Mediana	6451.7314

Tabla 3.6: Estadísticas de las 20 corridas del experimento 1.

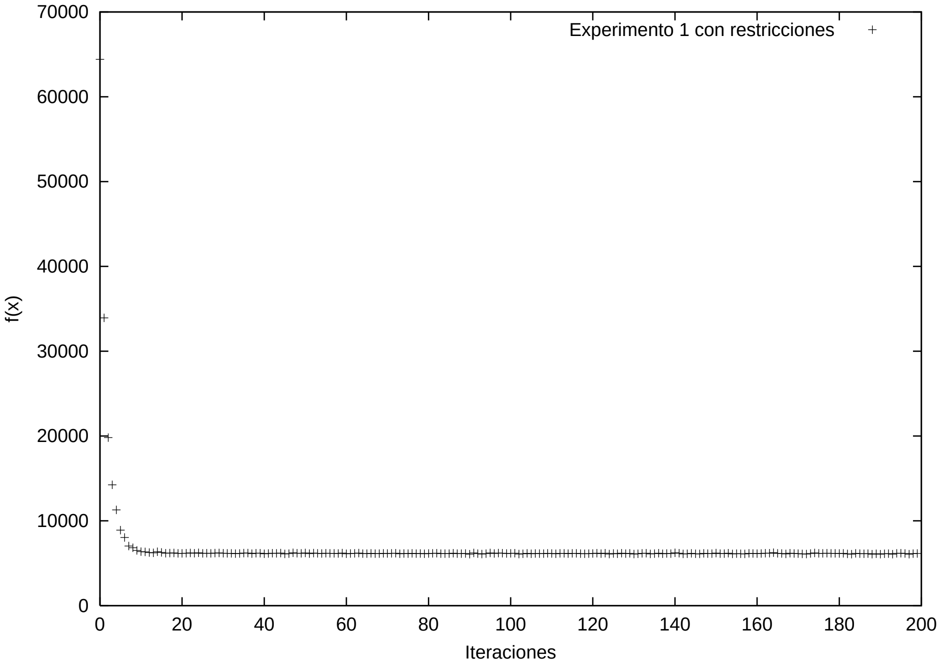

Figura 3.1: Gráfica de convergencia de la corrida 12 (Experimento 1).

3.2.2 Experimento 2

Este segundo experimento también es una función de optimización de ingeniería, cuyo propósito es el diseño de una viga soldada. La función es la siguiente:

$Minimizar:$

$$F(X) = 1.10471x_1^2x_2 + 0.04811x_3x_4(14.0 + x_2) \tag{3.6}$$

$Sujeto\ a:$

$$\begin{aligned} g1(X) &= \tau(X) - \tau_{max} \leq 0 && (3.7)\\ g2(X) &= \sigma(X) - \sigma_{max} \leq 0 && (3.8)\\ g3(X) &= x_1 - x_4 \leq 0 && (3.9)\\ g4(X) &= 0.10471x_1^2 + 0.04811x_3x_4(14.0 + x_2) - 5.0 \leq 0 && (3.10)\\ g5(X) &= 0.125 - x_1 \leq 0 && (3.11)\\ g6(X) &= \delta(X) - \delta_{max} \leq 0 && (3.12)\\ g7(X) &= P - P_c(X) \leq 0 && (3.13) \end{aligned}$$

$donde:$

$$\begin{aligned} \tau(X) &= \sqrt{(\tau')^2 + 2\tau'\tau''\frac{x_2}{2R} + (\tau'')^2} && (3.14)\\ \tau'(X) &= \frac{P}{\sqrt{2}x_1x_2} && (3.15)\\ \tau''(X) &= \frac{MR}{J} && (3.16)\\ M &= P\left(L + \frac{x_2}{2}\right) && (3.17)\\ R &= \sqrt{\frac{x_2^2}{4} + \left(\frac{x_1 + x_3}{2}\right)^2} && (3.18)\\ J &= 2\left\{\sqrt{2}x_1x_2\left[\frac{x_2^2}{12} + \left(\frac{x_1 + x_3}{2}\right)^2\right]\right\} && (3.19)\\ \sigma(X) &= \frac{6PL}{x_4x_3^2} && (3.20)\\ \delta(X) &= \frac{4PL^3}{Ex_3^3x_4} && (3.21)\\ Pc(X) &= \frac{4.013E\sqrt{\frac{x_3^2x_4^6}{36}}}{L^2}\left(1 - \frac{x_3}{2L}\sqrt{\frac{E}{4G}}\right) && (3.22) \end{aligned}$$

$$P = 6000\ lb,\ L = 14\ in,\ E = 30 \times 10^6\ psi,\ G = 12 \times 10^6\ psi$$

$$\tau_{max} = 13,600\ psi,\ \sigma_{max} = 30,000\ psi,\ \delta_{max} = 0.25\ in$$

$y:$

$$0.1 \leq x_1, x_4 \leq 2.0 \text{ y } 0.1 \leq x_2, x_3 \leq 10.0$$

Variables de Diseño	**Mejor solución encontrada ADRANS[37]**	**Deb[9]**	**APPROX[37]**
$x_1(h)$	0.239300	0.248900	0.244400
$x_2(l)$	6.328000	6.173000	6.218900
$x_3(t)$	8.369500	8.178900	8.291500
$x_4(b)$	0.245800	0.253300	0.244400
$g1(X)$	-5734.961126	-5758.603777	-5743.502027
$g2(X)$	-728.186430	-255.576901	-4.015209
$g3(X)$	-0.006500	-0.004400	0.000000
$g4(X)$	-2.982081	-2.982866	-3.022561
$g5(X)$	-0.114300	-0.123900	-0.119400
$g6(X)$	-0.234767	-0.234160	-0.234243
$g7(X)$	-3717.298290	-4465.270928	-3490.469418
$f(X)$	**2.41223664**	**2.43311600**	**2.38154338**

Tabla 3.7: Mejores Soluciones reportadas en la literatura para la función del experimento 2. **Parte I**

Esta función, así como los mejores resultados reportados en la literatura (tablas 3.7, 3.8, 3.9 y3.10), se encuentran en [3].

Como podemos observar, la función cuenta con cuatro variables de decisión y siete restricciones.

En la tabla 3.11 se muestran los resultados de veinte corridas, ejecutadas con los parámetros que se encuentran en la tabla 3.12 para todas las corridas; es decir, en la tabla 3.11 se encuentran resumidos los mejores resultados obtenidos en cada una de las veinte corridas y además observamos los valores de cada una de las variables, para cada una de las corridas y el valor que la función toma. En esa misma tabla también encontramos un campo que nos indica si el resultado es una solución factible o no; esto último está representado mediante el símbolo de pertenencia.

Además, en la tabla 3.13, podemos observar los valores obtenidos para cada una de las restricciones impuestas por este problema de ingeniería. Y al igual que en la tabla 3.11, se muestra un campo en el que se indica si la solución generada en esa corrida fue factible o no.

Finalmente, en la tabla 3.14 se muestran algunos datos estadísticos generados con los resultados de las veinte corridas. Y en la figura 3.2 vemos la gráfica de convergencia correspondiente a la corrida número 20, que es la solución más cercana a la mediana.

Variables de Diseño	**Mejor solución encontrada** Ragsdell[30]	DAVID[37]	MEMGRD[37]
$x_1(h)$	0.245500	0.243400	0.244100
$x_2(l)$	6.196000	6.255200	6.227300
$x_3(t)$	8.273000	8.291500	8.292700
$x_4(b)$	0.245500	0.244400	0.244400
$g1(X)$	-5743.826517	-5744.077735	-5742.345160
$g2(X)$	-4.715097	-4.015209	-12.695750
$g3(X)$	0.000000	-0.001000	-0.000300
$g4(X)$	-3.020289	-3.019074	-3.021472
$g5(X)$	-0.120500	-0.118400	-0.119100
$g6(X)$	-0.234208	-0.234243	-0.234250
$g7(X)$	-3604.275002	-3490.469418	-3491.423041
$f(X)$	**2.38593732**	**2.38410685**	**2.38219383**

Tabla 3.8: Mejores Soluciones reportadas en la literatura para la función del experimento 2. **Parte II**

Variables de Diseño	**Mejor solución encontrada** SEEK1*[37]	SEEK3[37]	SIMPLX*[37]
$x_1(h)$	0.391800	0.245400	0.279200
$x_2(l)$	4.615400	6.214800	5.625600
$x_3(t)$	6.230900	8.254600	7.751200
$x_4(b)$	0.432700	0.246600	0.279600
$g1(X)$	-5951.757624	-5743.540457	-5795.870533
$g2(X)$	1.438501	-5.239546	2.364539
$g3(X)$	-0.040900	-0.001200	-0.000400
$g4(X)$	-2.569325	-3.014019	-2.945561
$g5(X)$	-0.266800	-0.120400	-0.154200
$g6(X)$	-0.229028	-0.234173	-0.233141
$g7(X)$	-36585.346851	-3718.888760	-7548.672647
$f(X)$	**3.19728523**	**2.39312654**	**2.53072583**

Tabla 3.9: Mejores Soluciones reportadas en la literatura para la función del experimento 2. *Solución no factible. **Parte III**

Variables de Diseño	**Mejor solución encontrada RANDOM[37]**	**Coello[4]**
$x_1(h)$	0.457500	0.208800
$x_2(l)$	4.731300	3.420500
$x_3(t)$	5.085300	8.997500
$x_4(b)$	0.660000	0.210000
$g1(X)$	-6056.720602	-0.337812
$g2(X)$	-470.680167	-353.902604
$g3(X)$	-0.202500	-0.001200
$g4(X)$	-1.953512	-3.411865
$g5(X)$	-0.332500	-0.083800
$g6(X)$	-0.224708	-0.235649
$g7(X)$	-122179.185152	-363.232384
$f(X)$	**4.11855504**	**1.74830941**

Tabla 3.10: Mejores Soluciones reportadas en la literatura para la función del experimento 2. **Parte IV**

Corrida	x1	x2	x3	x4	f(x)	Factible
1	0.17	5.32	7.44	0.30	2.28	$\in$
2	0.68	1.41	4.91	0.70	3.25	$\in$
3	0.37	2.31	6.47	0.40	2.39	$\in$
4	0.42	2.08	6.09	0.45	2.55	$\in$
5	0.68	1.43	4.89	0.70	3.27	$\in$
6	0.31	2.72	6.77	0.37	2.29	$\notin$
7	0.35	2.34	6.90	0.35	2.23	$\in$
8	0.23	3.17	8.55	0.23	1.81	$\in$
9	0.25	3.05	8.15	0.25	1.90	$\in$
10	0.20	3.93	8.08	0.26	1.97	$\in$
11	0.42	3.07	4.18	0.96	3.91	$\in$
12	0.65	1.45	5.10	0.65	3.12	$\in$
13	0.29	3.01	6.84	0.36	2.29	$\in$
14	0.30	3.61	5.53	0.55	2.93	$\in$
15	0.58	1.63	5.30	0.60	2.98	$\notin$
16	0.17	6.87	6.31	0.42	2.88	$\in$
17	0.21	3.36	8.93	0.21	1.76	$\in$
18	0.36	2.43	6.42	0.41	2.41	$\in$
19	0.45	2.90	4.07	1.03	4.06	$\in$
20	0.41	2.05	6.39	0.41	2.42	$\in$

Tabla 3.11: Soluciones encontradas para el experimento 2 en 20 corridas.

Tamaño del cúmulo	Número de iteraciones	W	$C1$	$C2$
120	200	0.2	1.2	1.2

Tabla 3.12: Parámetros utilizados en el experimento 2.

Corrida	g1	g2	g3	g4	g5	g6	g7	Factible
1	-17.03	-0.28	-0.13	-2.90	-0.05	-0.23	-10773.37	$\in$
2	-0.18	-4.04	-0.02	-2.42	-0.55	-0.22	-140243.12	$\in$
3	-1.36	-20.94	-0.03	-2.95	-0.24	-0.23	-29082.64	$\in$
4	-63.90	-0.35	-0.03	-2.85	-0.30	-0.23	-42084.64	$\in$
5	-101.03	-0.69	-0.03	-2.40	-0.55	-0.22	-143021.25	$\in$
6	-15.12	0.39	-0.05	-2.99	-0.19	-0.23	-21613.99	$\notin$
7	-0.11	-40.32	-0.01	-3.07	-0.22	-0.23	-19024.93	$\in$
8	-1.07	-107.82	-0.00	-3.37	-0.11	-0.23	-2125.22	$\in$
9	-96.29	-20.32	-0.00	-3.30	-0.12	-0.23	-4417.13	$\in$
10	-0.79	-0.00	-0.05	-3.20	-0.08	-0.23	-4844.59	$\in$
11	-7.86	-30.45	-0.54	-1.68	-0.30	-0.22	-328933.14	$\in$
12	-0.31	-85.81	-0.00	-2.50	-0.52	-0.22	-115378.49	$\in$
13	-0.01	-0.03	-0.07	-2.98	-0.16	-0.23	-20220.40	$\in$
14	-0.15	-1.37	-0.25	-2.42	-0.17	-0.23	-73083.88	$\in$
15	-90.83	3.00	-0.02	-2.58	-0.45	-0.23	-92654.43	$\notin$
16	-0.01	-4.45	-0.26	-2.32	-0.04	-0.23	-33858.34	$\in$
17	-0.01	-372.42	-0.00	-3.41	-0.09	-0.24	-623.97	$\in$
18	-0.00	-4.38	-0.05	-2.92	-0.23	-0.23	-30553.74	$\in$
19	-81.74	-415.11	-0.57	-1.58	-0.33	-0.22	-393023.63	$\in$
20	-7.62	-2.38	-0.00	-2.95	-0.29	-0.23	-31433.49	$\in$

Tabla 3.13: Valores de las restricciones para el experimento 2 de las 20 corridas realizadas.

Mejor solución	1.7584
Peor solución	2.9769
Media	2.6349
Varianza	0.4234
Desv. Est.	0.6507
Mediana	2.4144

Tabla 3.14: Estadísticas de las 20 corridas del experimento 2.

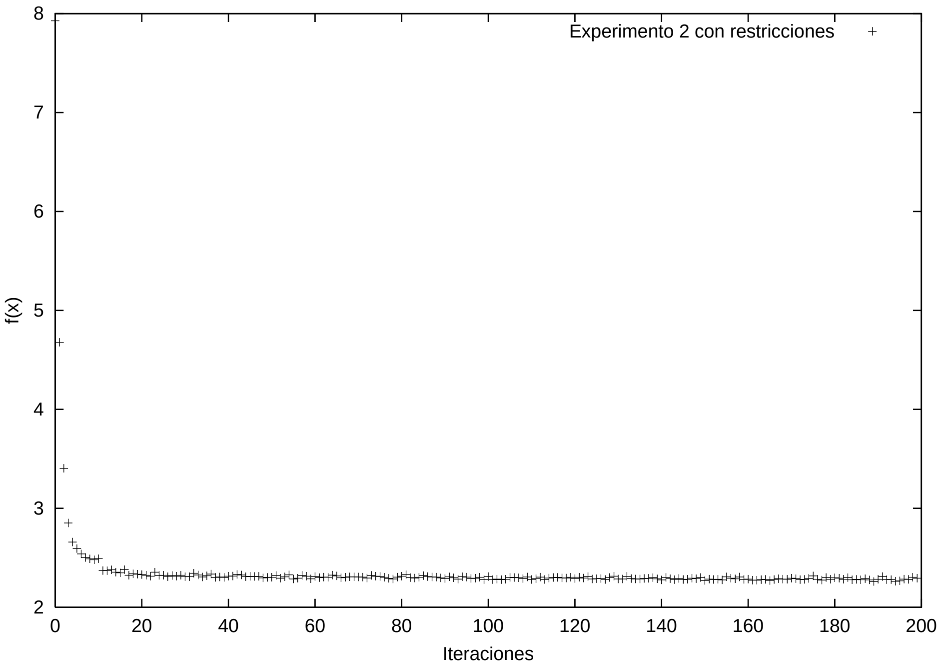

Figura 3.2: Gráfica de convergencia de la corrida 20 (Experimento 2).

3.2.3 Experimento 3

El tercer experimento intenta minimizar la siguiente función:
$Minimizar:$

$$f(x) = 5.3578547x_3^2 + 0.8356891x_1x_5 + 37.293239x_1 - 40792.141 \tag{3.23}$$

$Sujeto\ a:$

$$\begin{aligned}
g1(x) &= 85.334407 + 0.0056858x_2x_5 \\
&\quad +0.0006262x_1x_4 - 0.0022053x_3x_5 - 92 \leq 0 && (3.24)\\
g2(x) &= -85.334407 - 0.0056858x_2x_5 \\
&\quad -0.0006262x_1x_4 + 0.0022053x_3x_5 \leq 0 && (3.25)\\
g3(x) &= 80.51249 + 0.0071317x_2x_5 \\
&\quad +0.0029955x_1x_2 + 0.0021813x_3^2 - 110 \leq 0 && (3.26)\\
g4(x) &= -80.51249 - 0.0071317x_2x_5 \\
&\quad -0.0029955x_1x_2 - 0.0021813x_3^2 + 90 \leq 0 && (3.27)\\
g5(x) &= 9.300961 + 0.0047026x_3x_5 \\
&\quad +0.0012547x_1x_3 + 0.0019085x_3x_4 - 25 \leq 0 && (3.28)\\
g6(x) &= -9.300961 - 0.0047026x_3x_5 \\
&\quad -0.0012547x_1x_3 - 0.0019085x_3x_4 + 20 \leq 0 && (3.29)
\end{aligned}$$

$donde:$

$$78 \leq x_1 \leq 102,\ 33 \leq x_2 \leq 45,\ \text{y}\ 27 \leq x_i \leq 45\ (i = 3, 4, 5)$$

Esta función, así como los mejores resultados reportados en la literatura (tabla 3.15), pueden ser encontrados en [33], en donde recibe el nombre de $g04$.

$Variables$					$Solución$
x_1	x_2	x_3	x_4	x_5	$f(\vec{x})$
78	33	29.995256025682	45	36.775812905788	**-30665.539**

Tabla 3.15: Mejor Solución reportada en la literatura para la función del experimento 3.

Como podemos observar, la función cuenta con cinco variables de decisión y seis restricciones.

En la tabla 3.16 se muestran los resultados de veinte corridas, ejecutadas con los parámetros que se encuentran en la tabla 3.17 para todas las corridas; es decir, en la tabla 3.16 se encuentran resumidos los mejores resultados obtenidos en cada una de las veinte corridas y además observamos los valores de cada una de las variables, para cada una de las corridas y el valor que la función toma. En esa misma tabla también encontramos un campo que nos indica si el resultado es una solución factible o no; esto último está representado mediante el símbolo de pertenencia.

Corrida	x1	x2	x3	x4	x5	f(x)	Factible
1	79.37	33.00	30.30	45.00	35.64	-30548.53	$\in$
2	78.00	38.65	32.79	45.00	30.30	-30145.80	$\in$
3	78.23	35.06	31.07	44.97	34.10	-30473.33	$\in$
4	81.36	33.22	30.12	45.00	35.57	-30479.51	$\in$
5	78.00	33.17	30.09	44.99	36.54	-30650.63	$\in$
6	78.63	33.01	30.04	36.20	40.07	-30392.19	$\in$
7	78.00	35.24	31.17	43.80	34.40	-30434.46	$\in$
8	78.00	33.11	30.06	40.36	38.50	-30532.59	$\in$
9	78.00	36.25	31.76	39.07	34.97	-30199.42	$\in$
10	78.00	33.00	30.00	36.97	40.01	-30451.74	$\in$
11	80.13	33.13	30.15	45.00	35.83	-30535.19	$\in$
12	78.00	33.04	30.01	45.00	36.73	-30662.45	$\in$
13	78.87	34.71	30.89	43.80	34.84	-30442.52	$\in$
14	78.04	33.34	30.18	44.48	36.52	-30621.13	$\in$
15	78.00	33.28	30.15	43.26	37.10	-30595.54	$\in$
16	78.00	33.42	30.36	45.00	35.86	-30606.37	$\in$
17	78.00	33.58	30.31	43.19	36.72	-30567.06	$\in$
18	78.00	37.25	32.14	44.96	31.72	-30279.99	$\in$
19	78.00	37.27	32.17	44.73	31.75	-30267.61	$\in$
20	78.00	33.05	30.02	44.99	36.71	-30661.03	$\in$

Tabla 3.16: Soluciones encontradas para el experimento 3 en 20 corridas.

Tamaño del cúmulo	Número de iteraciones	W	$C1$	$C2$
120	200	0.2	1.2	1.2

Tabla 3.17: Parámetros utilizados en el experimento 3.

Corrida	g1	g2	g3	g4	g5	g6	Factible
1	-0.12	-91.88	-11.25	-8.75	-5.00	-0.00	$\in$
2	-0.00	-92.00	-9.76	-10.24	-5.00	-0.00	$\in$
3	-0.00	-92.00	-10.64	-9.36	-5.00	-0.00	$\in$
4	-0.02	-91.98	-10.99	-9.01	-5.00	-0.00	$\in$
5	-0.00	-92.00	-11.12	-8.88	-5.00	-0.00	$\in$
6	-0.02	-91.98	-10.31	-9.69	-5.00	-0.00	$\in$
7	-0.00	-92.00	-10.49	-9.51	-5.00	-0.00	$\in$
8	-0.00	-92.00	-10.69	-9.31	-5.00	-0.00	$\in$
9	-0.00	-92.00	-9.78	-10.22	-5.00	-0.00	$\in$
10	-0.00	-92.00	-10.40	-9.60	-5.00	-0.00	$\in$
11	-0.04	-91.96	-11.09	-8.91	-5.00	-0.00	$\in$
12	-0.00	-92.00	-11.15	-8.85	-5.00	-0.00	$\in$
13	-0.00	-92.00	-10.58	-9.42	-5.00	-0.00	$\in$
14	-0.00	-92.00	-11.03	-8.97	-5.00	-0.00	$\in$
15	-0.00	-92.00	-10.93	-9.07	-5.00	-0.00	$\in$
16	-0.06	-91.94	-11.12	-8.88	-5.00	-0.00	$\in$
17	-0.00	-92.00	-10.84	-9.16	-5.00	-0.00	$\in$
18	-0.00	-92.00	-10.11	-9.89	-5.00	-0.00	$\in$
19	-0.01	-91.99	-10.08	-9.92	-5.00	-0.00	$\in$
20	-0.00	-92.00	-11.15	-8.85	-5.00	-0.00	$\in$

Tabla 3.18: Valores de las restricciones para el experimento 3 de las 20 corridas realizadas.

Además, en la tabla 3.18, podemos observar los valores obtenidos para cada una de las restricciones impuestas por este problema de ingeniería. Y al igual que en la tabla 3.16, se muestra un campo en el que se indica si la solución generada en esa corrida fue factible o no.

Finalmente, en la tabla 3.19 se muestran algunos datos estadísticos generados con los resultados de las veinte corridas. Y en la figura 3.3 vemos la representación de la gráfica de convergencia a la corrida número 10, correspondiente a la solución más cercana a la mediana.

Mejor solución	-30662.4470
Peor solución	-30145.7974
Media	-30477.3541
Varianza	23679.8670
Desv. Est.	153.8826
Mediana	-30506.0480

Tabla 3.19: Estadísticas de las 20 corridas del experimento 3.

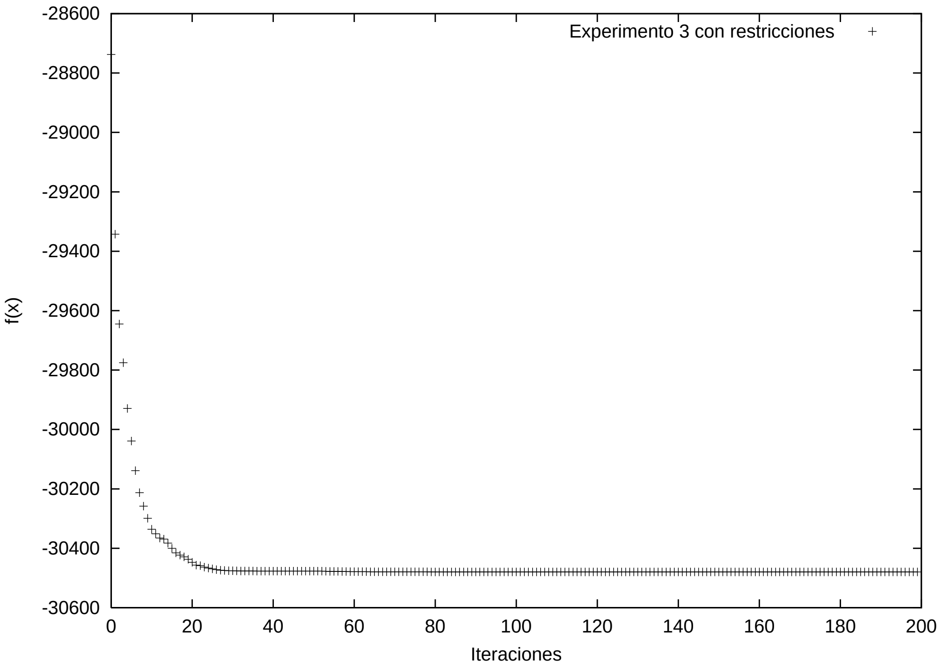

Figura 3.3: Gráfica de convergencia de la corrida 4 (Experimento 3).

3.2.4 Experimento 4

El cuarto experimento intenta minimizar la siguiente función:

Minimizar :

$$f(x) = (x_1 - 10)^3 + (x_2 - 20)^3 \tag{3.30}$$

Sujeto a :

$$g1(x) = -(x_1 - 5)^2 - (x_2 - 5)^2 + 100 \leq 0 \tag{3.31}$$

$$g2(x) = (x_1 - 6)^2 + (x_2 - 5)^2 - 82.81 \leq 0 \tag{3.32}$$

donde :

$$13 \leq x_1 \leq 100 \text{ y } 0 \leq x_2 \leq 100$$

Esta función, así como los mejores resultados reportados en la literatura (tabla 3.20), pueden ser encontrados en [33], en donde recibe el nombre de $g06$.

Variables		*Solución*
x_1	x_2	$f(\vec{x})$
14.095	0.84296	**-6961.81388**

Tabla 3.20: Mejor Solución reportada en la literatura para la función del experimento 4.

Como podemos observar, la función cuenta con dos variables de decisión y dos restricciones.

En la tabla 3.21 se muestran los resultados de veinte corridas, ejecutadas con los parámetros que se encuentran en la tabla 3.22 para todas las corridas; es decir, en la tabla 3.21 se encuentran resumidos los mejores resultados obtenidos en cada una de las veinte corridas y además observamos los valores de cada una de las variables, para cada una de las corridas y el valor que la función toma. En esa misma tabla también encontramos un campo que nos indica si el resultado es una solución factible o no; esto último está representado mediante el símbolo de pertenencia.

Además, en la tabla 3.23, podemos observar los valores obtenidos para cada una de las restricciones impuestas por este problema de ingeniería. Y al igual que en la tabla 3.21, se muestra un campo en el que se indica si la solución generada en esa corrida fue factible o no.

Finalmente, en la tabla 3.24 se muestran algunos datos estadísticos generados con los resultados de las veinte corridas. Y en la figura 3.4 vemos la representación de la gráfica de convergencia a la corrida número 4, correspondiente a la solución más cercana a la mediana.

Corrida	x1	x2	f(x)	Factible
1	14.37	1.43	-6325.17	$\in$
2	14.10	0.85	-6952.65	$\in$
3	14.14	0.93	-6860.24	$\in$
4	14.19	1.05	-6728.81	$\in$
5	14.10	0.85	-6948.85	$\in$
6	14.11	0.86	-6939.12	$\notin$
7	14.10	0.86	-6941.30	$\in$
8	14.31	1.29	-6471.42	$\in$
9	14.26	1.18	-6586.90	$\in$
10	14.25	1.16	-6609.19	$\in$
11	14.10	0.85	-6951.11	$\notin$
12	14.12	0.89	-6905.99	$\in$
13	14.33	1.34	-6412.36	$\in$
14	14.12	0.89	-6906.57	$\in$
15	14.10	0.84	-6960.74	$\in$
16	14.25	1.16	-6606.52	$\in$
17	14.18	1.00	-6783.35	$\in$
18	14.11	0.86	-6937.19	$\in$
19	14.75	2.52	-5236.80	$\in$
20	14.10	0.86	-6943.10	$\in$

Tabla 3.21: Soluciones encontradas para el experimento 4 en 20 corridas.

Tamaño del cúmulo	Número de iteraciones	W	$C1$	$C2$
120	200	0.2	1.2	1.2

Tabla 3.22: Parámetros utilizados en el experimento 4.

Corrida	g1	g2	Factible
1	-0.55	-0.00	$\in$
2	-0.01	-0.00	$\in$
3	-0.09	-0.00	$\in$
4	-0.03	-0.16	$\in$
5	-0.01	-0.00	$\in$
6	-0.02	0.00	$\notin$
7	-0.01	-0.01	$\in$
8	-0.43	-0.00	$\in$
9	-0.28	-0.04	$\in$
10	-0.31	-0.00	$\in$
11	-0.01	0.00	$\notin$
12	-0.05	-0.00	$\in$
13	-0.48	-0.00	$\in$
14	-0.05	-0.00	$\in$
15	-0.00	-0.00	$\in$
16	-0.31	-0.00	$\in$
17	-0.16	-0.00	$\in$
18	-0.02	-0.00	$\in$
19	-1.26	-0.06	$\in$
20	-0.02	-0.00	$\in$

Tabla 3.23: Valores de las restricciones para el experimento 4 de las 20 corridas realizadas.

Mejor solución	-6960.7441
Peor solución	-6939.1228
Media	-6700.3703
Varianza	160648.1628
Desv. Est.	400.8094
Mediana	-6883.1175

Tabla 3.24: Estadísticas de las 20 corridas del experimento 4.

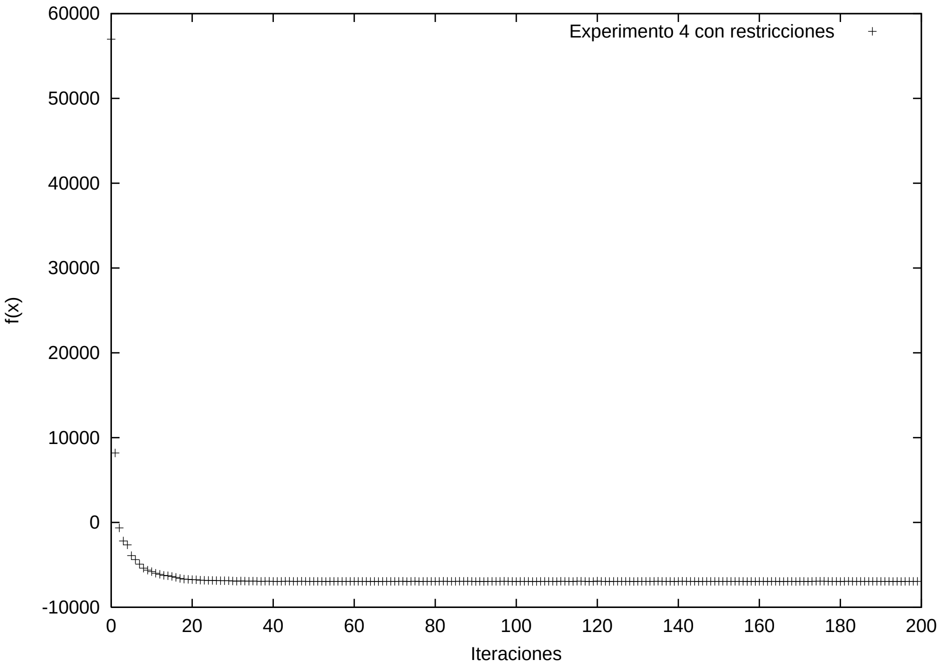

Figura 3.4: Gráfica de convergencia de la corrida 12 (Experimento 4).

3.2.5 Experimento 5

El quinto experimento intenta minimizar la siguiente función:
Minimizar :

$$f(x) = \frac{\sin^3(2\pi x_1)\sin(2\pi x_2)}{x_1^3(x_1 + x_2)} \tag{3.33}$$

Sujeto a :

$$g1(x) = x_1^2 - x_2 + 1 \leq 0 \tag{3.34}$$
$$g2(x) = 1 - x_1 + (x_2 - 4)^2 \leq 0 \tag{3.35}$$

donde :

$$0 \leq x_1 \leq 10 \text{ y } 0 \leq x_2 \leq 10$$

Esta función, así como los mejores resultados reportados en la literatura (tabla 3.25), pueden ser encontrados en [33], en donde recibe el nombre de $g08$.

Variables		*Solución*
x_1	x_2	$f(\vec{x})$
1.2279713	4.2453733	**0.095825**

Tabla 3.25: Mejor Solución reportada en la literatura para la función del experimento 5.

Como podemos observar, la función cuenta con dos variables de decisión y dos restricciones.

En la tabla 3.26 se muestran los resultados de veinte corridas, ejecutadas con los parámetros que se encuentran en la tabla 3.27 para todas las corridas; es decir, en la tabla 3.26 se encuentran resumidos los mejores resultados obtenidos en cada una de las veinte corridas y además observamos los valores de cada una de las variables, para cada una de las corridas y el valor que la función toma. En esa misma tabla también encontramos un campo que nos indica si el resultado es una solución factible o no; esto último está representado mediante el símbolo de pertenencia.

Además, en la tabla 3.28, podemos observar los valores obtenidos para cada una de las restricciones impuestas por este problema de ingeniería. Y al igual que en la tabla 3.26, se muestra un campo en el que se indica si la solución generada en esa corrida fue factible o no.

Finalmente, en la tabla 3.29 se muestran algunos datos estadísticos generados con los resultados de las veinte corridas. Y en la figura 3.5 vemos la representación de la gráfica de convergencia a la corrida número 4, correspondiente a la solución más cercana a la mediana.

Corrida	x1	x2	f(x)	Factible
1	1.23	3.74	-0.11	$\in$
2	1.23	3.74	-0.11	$\in$
3	1.23	3.74	-0.11	$\in$
4	1.23	3.74	-0.11	$\in$
5	1.23	3.74	-0.11	$\in$
6	1.23	3.74	-0.11	$\in$
7	1.23	3.74	-0.11	$\in$
8	1.23	3.74	-0.11	$\in$
9	1.23	3.74	-0.11	$\in$
10	1.23	3.74	-0.11	$\in$
11	1.23	3.74	-0.11	$\in$
12	1.23	3.74	-0.11	$\in$
13	1.23	3.74	-0.11	$\in$
14	1.23	3.74	-0.11	$\in$
15	1.23	3.74	-0.11	$\in$
16	1.23	3.74	-0.11	$\in$
17	1.23	3.74	-0.11	$\in$
18	1.23	3.74	-0.11	$\in$
19	1.23	3.74	-0.11	$\in$
20	1.23	3.74	-0.11	$\in$

Tabla 3.26: Soluciones encontradas para el experimento 5 en 20 corridas.

Tamaño del cúmulo	Número de iteraciones	W	$C1$	$C2$
120	200	0.2	1.2	1.2

Tabla 3.27: Parámetros utilizados en el experimento 5.

Corrida	g1	g2	Factible
1	-1.24	-0.16	$\in$
2	-1.24	-0.16	$\in$
3	-1.24	-0.16	$\in$
4	-1.24	-0.16	$\in$
5	-1.24	-0.16	$\in$
6	-1.24	-0.16	$\in$
7	-1.24	-0.16	$\in$
8	-1.24	-0.16	$\in$
9	-1.24	-0.16	$\in$
10	-1.24	-0.16	$\in$
11	-1.24	-0.16	$\in$
12	-1.24	-0.16	$\in$
13	-1.24	-0.16	$\in$
14	-1.24	-0.16	$\in$
15	-1.24	-0.16	$\in$
16	-1.24	-0.16	$\in$
17	-1.24	-0.16	$\in$
18	-1.24	-0.16	$\in$
19	-1.24	-0.16	$\in$
20	-1.24	-0.16	$\in$

Tabla 3.28: Valores de las restricciones para el experimento 5 de las 20 corridas realizadas.

Mejor solución	-0.1055
Peor solución	-0.1055
Media	-0.1055
Varianza	0.0000
Desv. Est.	0.0000
Mediana	-0.1055

Tabla 3.29: Estadísticas de las 20 corridas del experimento 5.

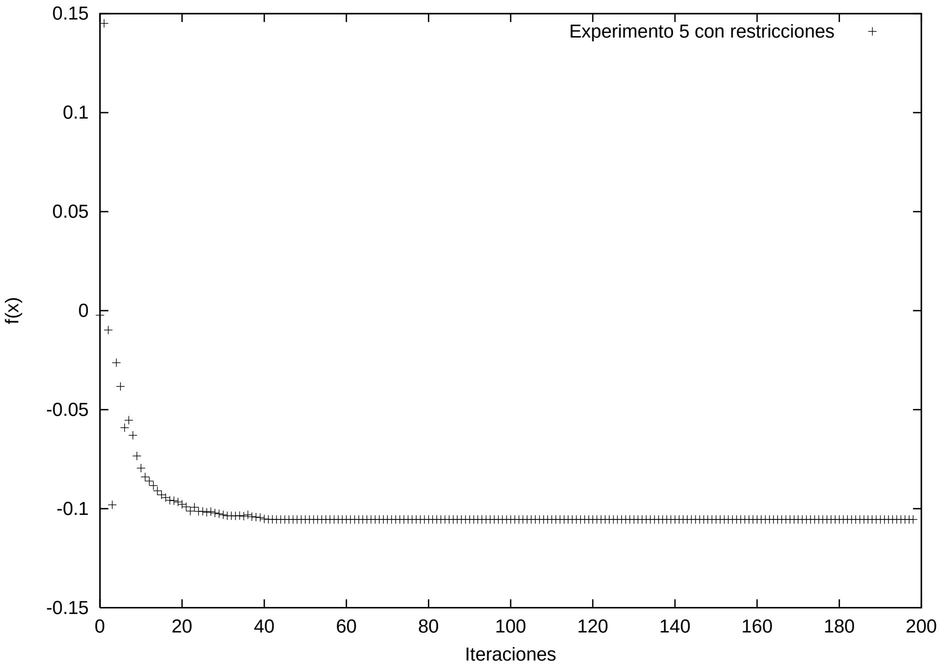

Figura 3.5: Gráfica de convergencia de la corrida 1 (Experimento 5).

3.3 Discusión de resultados de los experimentos

Podemos afirmar que la implementación de la PSO para manejo restricciones fue muy simple y que además dio buenos resultados. Esto último lo afirmamos una vez observados los experimentos que se realizaron a lo largo de este capítulo. En algunos casos los resultados fueron buenos en el sentido de que la mayoría de las soluciones fueron factibles, dado a que satisfacían a todas las funciones que las restringían, pero en algunas ocasiones, además se encontraban soluciones que incluso llegaban a mejorar a las reportadas en la literatura consultada.

Cabe mencionar la importancia del papel que juegan los coeficientes W, $C1$ y $C2$ en la fórmula de la velocidad. Aunque aquí estos coeficientes se utilizan fijos durante todos los experimentos, antes de realizar los experimentos de manera formal, tuvimos que ajustar empíricamente los valores que estos debían tomar, ya que notamos afectaban relevantemente el comportamiento del algoritmo. Después de haber observado el comportamiento de los algoritmos con varias funciones de prueba (algunas no mostradas aquí), decidimos usar los valores que utilizamos en estos experimentos ya que eran los que mayor estabilidad le daban al algoritmo en la mayoría de los casos, pero esto no necesariamente quiere decir que estos valores son los que mejor comportamiento muestran para cada uno de los experimentos individualmente. De hecho pudimos notar que en algunas funciones de prueba al cambiar los valores de los coeficientes que tomamos aquí pueden mejorar sus resultados, así como también algunas otras pueden mermar su desempeño notablemente.

Capítulo 4

Resolución de problemas multi-objetivo a través de PSO (MOPSO)

4.1 Introducción

Este capítulo muestra una propuesta para extender la heurística PSO y así resolver problemas de optimización multiobjetivo. Nuestro método usa el concepto de dominancia de Pareto para determinar la dirección de vuelo de una partícula y mantiene vectores no dominados encontrados previamente en un repositorio global que después es utilizado por otras partículas para guiar su vuelo.

4.2 Antecedentes

La mayoría de los problemas del mundo real son multiobjetivo por naturaleza, pero tienden a ser tratados como problemas de optimización de un solo objetivo, transformando la mayoría de los objetivos originales en restricciones adicionales. Los algoritmos evolutivos (AEs) son particularmente adaptables para resolver problemas de optimización multi-objetivo (MOP) porque son heurísticas basadas en poblaciones de soluciones y, por eso, permiten la generación de varios elementos del conjunto óptimo de Pareto en una sola ejecución. En contraste, las técnicas de programación matemática tradicionales tienden a generar un solo elemento del conjunto óptimo de Pareto en una sola ejecución. Adicionalmente, los AEs no son susceptibles a la forma o continuidad del frente de Pareto como lo son los métodos tradicionales.

El uso de AEs para resolver problemas multiobjetivo (un área llamada optimización evolutiva multiobjetivo) ha generado una considerable cantidad de investigación en los últimos años (ver por ejemplo [16, 2, 38]).

El desarrollo de una técnica (heurística) alterna para optimización multiobjetivo necesita contender con dos problemas principalmente: cómo generar vectores no dominados que se aproximen al frente de Pareto global del problema y cómo mantener diversidad para que podamos generar tantos vectores como nos sea posible. El método tradicional que maneja el primer problema ha sido el de usar un procedimiento llamado jerarquización de Pareto [19] el cual consiste en clasificar una población de individuos basados en el concepto de dominancia de Pareto. La jerarquización de Pareto es $O(kM^2)$, donde k es el número de funciones objetivo y M es el tamaño de la población. Adicionalmente, se requiere un mecanismo extra para preservar diversidad (suele adoptarse alguna forma de compartir aptitudes [12]). Esto generalmente implica el uso de otro proceso que es $O(M^2)$.

Conociendo las fuentes de la ineficiencia de las técnicas de optimización evolutiva multiobjetivo tradicionales, varios investigadores recientemente han enfocado sus esfuerzos en reducir el chequeo de no dominancia y en el desarrollo de métodos eficientes para mantener diversidad. Respecto al primer aspecto, el énfasis principal ha sido utilizar un archivo externo que almacena vectores no dominados encontrados durante el proceso evolutivo. Estos vectores son insertados de nuevo en la población en generaciones posteriores (esto puede ser visto como una forma de elitismo en el contexto de optimización multi-objetivo [21, 40]). Respecto al segundo aspecto, el énfasis principal ha sido usar técnicas de "clustering" [11] o métodos basados en posicionamiento geográfico de individuos en una malla adaptativa [26].

En este capítulo, presentamos una propuesta, llamada "Multi-Objective Particle Swarm Optimization" (MOPSO), la cual permite al algoritmo PSO ser capaz de manejar problemas de optimización multiobjetivo. La técnica es relativamente simple de implementar, se basa en el uso de una población, usa una memoria externa (llamada "repositorio") y un método basado en posicionamiento geográfico para mantener diversidad. MOPSO es validado en el siguiente capítulo usando diversas funciones de prueba estándares en la literatura especializada y comparado contra dos algoritmos OEM altamente competitivos: la *Pareto Archived Evolution Strategy* [26] (PAES) y el *Non-dominated Sorting Genetic Algorithm II* [11] (NSGA II).

4.3 Conceptos Básicos (Multi-Objetivo)

Definición 1 (MOP General): *Encontrar el vector* $\vec{x}^* = [x_1^*, x_2^*, \ldots, x_n^*]^T$ *que satisfaga las* m *restricciones de desigualdad:*

$$g_i(\vec{x}) \geq 0 \quad i = 1, 2, \ldots, m \tag{4.1}$$

las p *restricciones de igualdad*

$$h_i(\vec{x}) = 0 \quad i = 1, 2, \ldots, p \tag{4.2}$$

y optimice el vector función

$$\vec{f}(\vec{x}) = [f_1(\vec{x}), f_2(\vec{x}), \ldots, f_k(\vec{x})]^T \tag{4.3}$$

donde $\vec{x} = [x_1, x_2, \ldots, x_n]^T$ *es el vector de las variables de decisión.* □

Definición 2 (Optimalidad de Pareto): *Un punto* $\vec{x}^* \in \Omega$ *(*Ω *es la región factible) es un* **óptimo de Pareto** *si para cada* $\vec{x} \in \Omega$ *y* $I = \{1, 2, \ldots, k\}$ *tanto,*

$$\forall_{i \in I}(f_i(\vec{x}) = f_i(\vec{x}^*)) \tag{4.4}$$

o, hay al menos una $i \in I$ *tal que*

$$f_i(\vec{x}) > f_i(\vec{x}^*) \tag{4.5}$$

□

En palabras, esta definición dice que x es un óptimo de Pareto si no existe un vector x factible que haga decrementar algún criterio sin causar un aumento simultáneo en al menos algún otro criterio. El significado de la frase "óptimo de Pareto" es considerado respecto a todo el espacio de las variables de decisión a menos que se especifique lo contrario.

Definición 3 (Dominancia de Pareto): *Un vector* $\vec{u} = (u_1, \ldots, u_k)$ *se dice que domina a otro* $\vec{v} = (v_1, \ldots, v_k)$ *(denotado por* $\vec{u} \preceq \vec{v}$*) si y sólo si* u *es parcialmente menor que* v*, por ejemplo,* $\forall i \in \{1, \ldots, k\},\ u_i \leq v_i \wedge \exists i \in \{1, \ldots, k\} : u_i < v_i$*.* □

Definición 4 (Conjunto de Óptimos de Pareto): *Dado un MOP* $\vec{f}(x)$*, el conjunto de óptimos de Pareto (*$\mathcal{P}^*$*) es definido como:*

$$\mathcal{P}^* := \{x \in \Omega \mid \neg\exists\, x' \in \Omega\ \ \vec{f}(x') \preceq \vec{f}(x)\}. \tag{4.6}$$

□

Definición 5 (Frente de Pareto): *Para un MOP* $\vec{f}(x)$ *y un conjunto de óptimos de Pareto* $\mathcal{P}^*$ *dado, el frente de Pareto (*$\mathcal{PF}^*$*) es definido como:*

$$\mathcal{PF}^* := \{\vec{u} = \vec{f} = (f_1(x), \ldots, f_k(x)) \mid x \in \mathcal{P}^*\}. \tag{4.7}$$

□

4.4 Descripción del MOPSO

La optimización mediante cúmulos de partículas parece particularmente adaptable para optimización multiobjetivo debido principalmente a la alta velocidad de convergencia que el algoritmo presenta para problemas de optimización de un solo objetivo [25]. Nuestra hipótesis es que la extensión del PSO para manejar múltiples objetivos nos proveerá de un algoritmo muy veloz y además será relativamente fácil de implementar.

La analogía de la optimización mediante cúmulos de partículas con los algoritmos evolutivos hace evidente la noción de que usar un esquema como el de jerarquización de Pareto [19] podría ser una forma sencilla de extender la técnica para manejar problemas de optimización multiobjetivo. El registro histórico de las mejores soluciones encontradas por una partícula (por ejemplo, un individuo) podrían ser usadas para almacenar soluciones no dominadas generadas en el pasado (esto sería muy similar a la noción de elitismo usada en optimización evolutiva multiobjetivo [28]). El uso de mecanismos globales de atracción combinados con un archivo histórico de vectores no dominados encontrados anteriormente podría motivar convergencia a través de la generación de soluciones no dominadas globalmente.

Por eso, nuestra propuesta está basada en la idea de tener un repositorio global en el cual cada partícula depositará sus experiencias adquiridas después de cada ciclo de vuelo. Adicionalmente, las actualizaciones al repositorio se realizan considerando un sistema basado en posicionamiento geográfico definido en términos de los valores de la función objetivo de cada individuo. Esta técnica está inspirada en el archivo externo que es usado en la *Pareto Archive Evolution Strategy* (PAES) [26]. El repositorio previamente mencionado es usado por las partículas para identificar a un líder que guiará la búsqueda. Hemos implementado un mecanismo tal que cada partícula puede escoger a un guía distinto. Nuestro mecanismo está basado en la generación de hipercubos los cuales son producidos dividiendo el espacio de búsqueda explorado.

El algoritmo del MOPSO es el siguiente:

1. Inicializamos la población POP:
   (a) Para $i = 0$ hasta MAX /* MAX = número de partículas */
   (b) Inicializa $POP[i]$
2. Inicializamos la velocidad de cada partícula:
   (a) Para $i = 0$ hasta MAX
   (b) Inicializa $VEL[i]$
3. Evaluamos cada una de las partículas en POP.
4. Se guardan las posiciones de las partículas que representen vectores no dominados en el repositorio REP.
5. Se generan hipercubos del espacio de búsqueda explorado hasta el momento, y se localizan las partículas usando estos hipercubos como un sistema de coordenadas en donde la posición de cada partícula es definida de acuerdo a los valores de sus funciones objetivo.

6. Inicializamos la memoria de cada partícula con ella misma (esta memoria servirá como guía para viajar a través del espacio de búsqueda. Esta memoria es también almacenada en el repositorio):
   (a) Para $i = 0$, hasta MAX
   (b) $PBESTS[i] = POP[i]$
7. Mientras no se cumpla con el número de ciclos especificado:
   (a) Calculamos la velocidad de cada partícula[1] con la siguiente fórmula:

$$\begin{aligned} VEL[i] = W \times VEL[i] + R_1 \times (PBESTS[i] - POP[i]) + \\ R_2 \times (REP[h] - POP[i]) \end{aligned} \tag{4.8}$$

   En donde W (el peso de la inercia) toma un valor de 0.4; R_1 y R_2 son números aleatorios en el rango [0..1]; $PBESTS[i]$ es el mejor valor que esa partícula ha tomado[2]; $REP[h]$ es un valor que se toma del repositorio, en donde el índice h es seleccionado de la siguiente manera: a los hipercubos que contengan más de una partícula se les asigna una aptitud (la aptitud que se les otorga es del número de partículas que contengan dividido entre cualquier número x — nosotros usamos $x = 10$ —, esto con la finalidad de que los hipercubos con un mayor número de partículas tengan una aptitud menor que los que albergan a un número menor, y puede ser visto como una forma de compartir aptitudes [12]). Con base en esa aptitud se hace una ruleta para escoger al hipercubo de donde se tomará a la partícula. Una vez que se seleccionó a un hipercubo, la partícula que se selecciona dentro del hipercubo es tomada aleatoriamente; $POP[i]$ es el valor que tiene actualmente la partícula i.

   (b) Se calculan las nuevas posiciones de las partículas sumando la velocidad obtenida en el paso anterior:

$$POP[i] = POP[i] + VEL[i] \tag{4.9}$$

   (c) Se mantienen las partículas dentro del espacio. Esto es, en caso de que las partículas hayan salido del hipercubo en el que se encuentran realizando la búsqueda se reintegran de nuevo (evitando así generar soluciones que no residan en el espacio de búsqueda válido).
   (d) Se evalúan cada una de las partículas POP.

[1] Cada partícula cuenta con una dimensionalidad que puede ser distinta dependiendo del problema. Cuando decimos que calculamos la velocidad de una partícula, nos referimos a que calculamos una velocidad para cada una de sus dimensiones.

[2] Más adelante se explicará el criterio para tomar en cuenta cuál es el mejor valor que esa partícula ha tomado.

(e) Se va actualizando el repositorio REP junto con la representación geográfica que existe de las partículas en los hipercubos. La actualización consiste en insertar todas las posiciones actuales en el repositorio en caso de que se traten de posiciones no dominadas, e integrar su posición en el hipercubo correspondiente. En caso de que existan posiciones actuales que dominen a posiciones en el repositorio las posiciones en el repositorio son eliminadas y se actualiza igualmente el hiper espacio al que pertenecían. En el supuesto de que el repositorio esté a su máxima capacidad, las partículas que se retienen son aquellas que se encuentren en zonas menos pobladas del espacio de búsqueda, con el fin de mantener la diversidad de las partículas.

(f) Cuando la posición actual de la partícula sea mejor que la que contiene en su memoria, ésta se actualiza de la siguiente manera:

$$PBESTS[i] = POP[i] \tag{4.10}$$

El criterio utilizado para decidir qué posición retener en la memoria es simplemente aplicar dominancia de Pareto (por ejemplo, si la posición actual se encuentra dominada por la posición memorizada, se retiene la memorizada; en caso contrario, se actualiza la memoria por la posición actual; en caso de que no sea una mejor o peor posición se escoge a una de las dos posiciones aleatoriamente).

(g) Aumentamos el ciclo.

8. Fin del Mientras.

Capítulo 5

Resolución de problemas multi-objetivo a través de MOPSO (experimentos)

5.1 Introducción

En este capítulo mostramos algunos resultados que obtuvimos con la implementación del algoritmo propuesto en el capítulo anterior al compararlo contra algunas otras técnicas que representan el estado del arte en la actualidad. Además, discutiremos los resultados.

5.2 Comparación de resultados

Diversas funciones de prueba fueron tomadas de la literatura especializada para comparar nuestro método. Para comparar nuestros resultados de una forma cuantitativa usamos dos criterios: el tiempo promedio de ejecución del algoritmo (usando el mismo número de evaluaciones a la función objetivo), y la siguiente métrica definida en el espacio de las funciones objetivo por Zitzler et al. [40]:

$$M_1^* = \frac{1}{\mid Y' \mid} \sum_{\mathbf{d}' \in Y'} \min \left\{ \| \mathbf{d}' - \bar{\mathbf{d}} \|^* ; \bar{\mathbf{d}} \in \bar{Y} \right\} \tag{5.1}$$

donde: $Y', \bar{Y} \subseteq Y$ son los conjuntos de los vectores objetivo que corresponden a un conjunto de pares de vectores de decisión no dominados $X', \bar{X} \subseteq X$, respectivamente, y X corresponde a las variables de decisión del problema. Debe ser obvio que M_1^* da distancia promedio al conjunto óptimo de Pareto. Es por eso que debemos intentar minimizar este valor (ver [40] para más detalles). Para calcular M_1^*, generamos el frente de Pareto global para cada una de las funciones de prueba (esto es hecho mediante enumeración, usando una representación de grano grueso del espacio de variables de decisión), de tal forma que podamos medir la cercanía al conjunto de soluciones deseado.

Dado que el principal esfuerzo de este método ha sido el de incrementar eficiencia, comparamos nuestro MOPSO contra dos algoritmos muy veloces que representan el estado del arte en optimización evolutiva multiobjetivo: NSGA-II [11] y PAES[1] [26].

En los siguientes ejemplos, el NSGA-II fue ejecutado usando un tamaño de población de 200 individuos, un porcentaje de cruza de 0.8, selección mediante torneo, y un procentaje de mutación de 1/vars, en donde vars = al número de variables de decisión del problema. En los siguientes ejemplos, PAES fue ejecutada usando una profundidad de 5, un tamaño de archivo de 200, y un porcentaje de mutación de $1/L$, donde L se refiere a la longitud de la cadena del cromosoma que codifica las variables de decisión.

MOPSO usó una población de 80 partículas en todas las funciones de prueba. Aunque cabe mencionar que esto se hizo con el fin de uniformizar el valor de este parámetro, ya que por ejemplo, en la primera y tercera función de prueba se han obtenido buenos resultados utilizando poblaciones de 40 individuos, y en la segunda función de prueba se han visto muy buenos resultados con una población de tan solo 20 partículas. Se uso un repositorio capaz de alojar a 200 individuos para todos los experimentos. El número de divisiones del espacio de la función objetivo fue de 30 para todas la funciones de prueba. Y el valor de la W en la función para calcular la velocidad tomó el valor de 0.4; cabe mencionar que este valor se escogió en base a un proceso empírico, el cual nos determino que con ese valor la técnica se comportaba eficientemente con todos los problemas.

Para permitir una comparación justa en tiempos de ejecución, todos los experimentos se llevaron a cabo en una PC con un procesador Pentium II a 333 MHz, 256 Mb en RAM y un disco duro de 20 Gbytes. Nuestra implementación se compiló utilizando GNU C ejecutado en la plataforma Linux Red Hat versión 7.1.

5.2.1 Función de Prueba 1

La primera función de prueba fue propuesta por Deb [8]:

$$\text{Min } f_1(x_1, x_2) = x_1 \tag{5.2}$$

$$\begin{aligned}\text{Min } f_2(x_1, x_2) \;=\; (1 + 10x_2) \times \Bigg[1 - \left(\frac{x_1}{1+10x_2}\right)^{\alpha} - \\ \frac{x_1}{1+10x_2}\sin(2\pi q x)\Bigg]\end{aligned} \tag{5.3}$$

[1]El código fuente del NSGA II y PAES (versiones de sus autores correspondientes) pueden ser descargadas del repositorio EMOO localizado en `http://www.lania.mx/~ccoello/EMOO/EMOOsoftware.html`

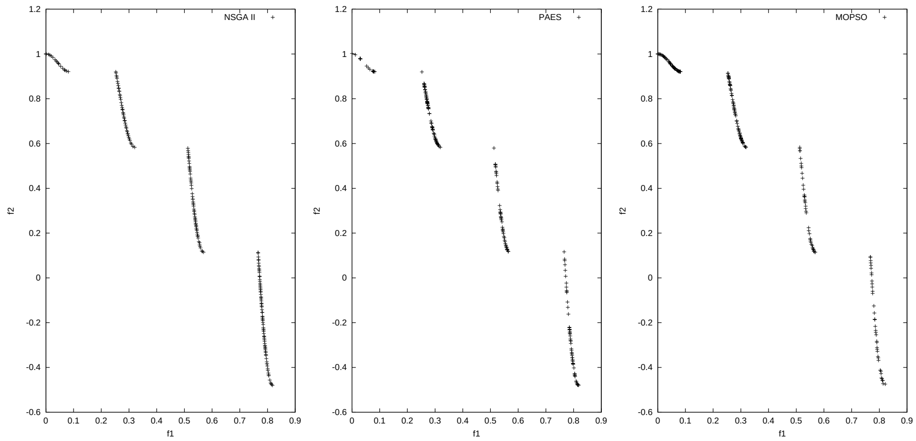

Figura 5.1: Frentes de Pareto producidos por el NSGA-II (izquierda), PAES (en medio), y MOPSO (derecha) para la primera función de prueba.

donde:

$$0 \leq x_1, x_2 \leq 1 \tag{5.4}$$

y $q = 4$, $\alpha = 2$. Este problema tiene un frente de Pareto desconectado y consistente en 4 curvas.

Las gráficas de la figura 5.1 muestran los frentes de Pareto producidos por el NSGA-II, PAES y MOPSO, respectivamente para la primera función de prueba. Para este ejemplo, los tres algoritmos realizaron 4000 evaluaciones de la función objetivo. Los tiempos promedio de ejecución de cada algoritmo (20 corridas) fueron los siguientes: 2.402 segundos para el NSGA-II, 2.08 para el PAES y sólo 0.076 segundos para el MOPSO. Los valores promedio de la métrica M_1^* fueron los siguientes: 0.002535573 para el NSGA-II (con una desviación estándar de 0.000138018), 0.002881265 para el PAES (con una desviación estándar de 0.002130444), y 0.002056797 para el MOPSO (con una desviación estándar de 0.000286163).

5.2.2 Función de Prueba 2

La segunda función de prueba fue propuesta por Schaffer [35]:

$$\text{Min } f_1(x) = \begin{cases} -x & \text{si } x \leq 1 \\ -2 + x & \text{si } 1 < x \leq 3 \\ 4 - x & \text{si } 3 < x \leq 4 \\ -4 + x & \text{si } x > 4 \end{cases} \tag{5.5}$$

$$\text{Min } f_2(x) = (x - 5)^2 \tag{5.6}$$

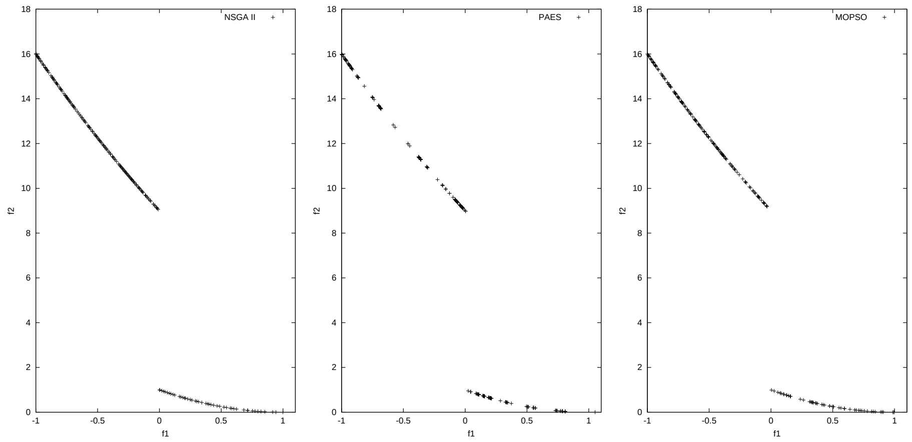

Figura 5.2: Frentes de Pareto producidos por el NSGA-II (izquierda), PAES (en medio), y MOPSO (derecha) para la segunda función de prueba.

donde: $-5 \leq x \leq 10$.

Este problema también tiene un frente de Pareto discontinuo. Las gráficas de la figura 5.2 muestran los frentes de Pareto producidos por el NSGA-II, PAES y MOPSO, respectivamente para la segunda función de prueba. Para este ejemplo, los tres algoritmos realizaron 1200 evaluaciones de la función objetivo. El promedio de los tiempos de ejecución de cada algoritmo (20 corridas) fueron los siguientes: 0.812 segundos para el NSGA-II, 0.339 segundos para el PAES y sólo 0.046 segundos para el MOPSO. Los valores promedio de la métrica M_1^* fueron los siguientes: 0.001594367 para el NSGA-II (con una desviación estándar de 0.000122178), 0.070003463 para el PAES (con una desviación estándar de 0.158081204), y 0.001473958 para el MOPSO (con una desviación estándar de 0.000201779).

5.2.3 Función de Prueba 3

El tercer ejemplo fue propuesto por Deb [8]:

$$\text{Minimizar } f_1(x_1, x_2) = x_1 \tag{5.7}$$

$$\text{Minimizar } f_2(x_1, x_2) = g(x_1, x_2) \cdot h(x_1, x_2) \tag{5.8}$$

donde:

$$g(x_1, x_2) = 11 + x_2^2 - 10 \cdot \cos(2\pi x_2) \tag{5.9}$$

$$h(x_1, x_2) = \begin{cases} 1 - \sqrt{\frac{f_1(x_1,x_2}{g(x_1,x_2}} & \text{si } f_1(x_1, x_2) \leq g(x_1, x_2) \\ 0 & \text{en otro caso} \end{cases} \tag{5.10}$$

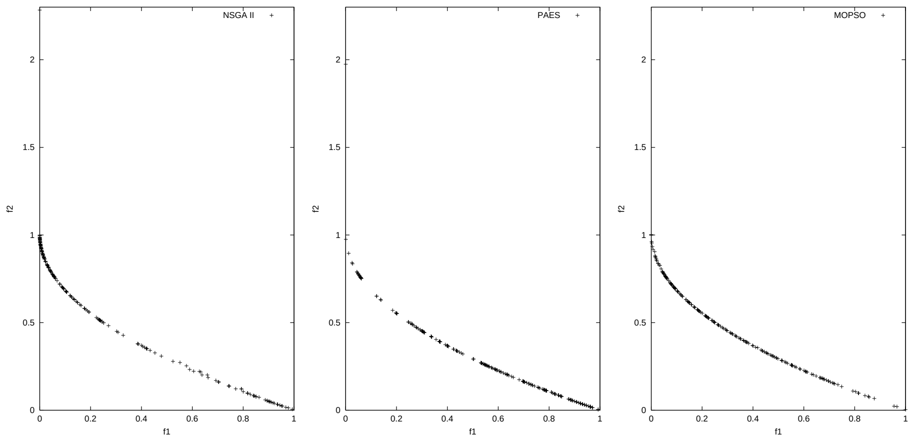

Figura 5.3: Frentes de Pareto producidos por el NSGA-II (izquierda), PAES (en medio), y MOPSO (derecha) para la tercera función de prueba.

y $0 \leq x_1 \leq 1$, $-30 \leq x_2 \leq 30$.

Este problema es multifrontal. Es decir, existen varios falsos frentes de Pareto (60 en este caso) a los cuales la población podría ser fácilmente atraída.

Las gráficas de la figura 5.3 muestran los frentes de Pareto producidos por el NSGA-II, PAES y MOPSO, respectivamente para la tercera función de prueba. Para este ejemplo, los tres algoritmos llevaron a cabo 3200 evaluaciones de la función objetivo. Los tiempos promedio de ejecución de cada algoritmo (20 corridas) son los siguientes: 2.1165 segundos para el NSGA-II, 1.641 segundos para el PAES y solo 0.098 segundos para el MOPSO. Los valores promedio para la métrica M_1^* fueron los siguientes: 0.094643769 para el NSGA-II (con una desviación estándar de 0.117607508), 0.259663944 para el PAES (con una desviación estándar de 0.573286406), y 0.001161128 para el MOPSO (con una desviación estándar de 0.000720476).

5.2.4 Función de Prueba 4

El cuarto ejemplo fue propuesto por Deb [8]:

$$\text{Minimizar } f_1(x_1, x_2) = x_1 \tag{5.11}$$

$$\text{Minimizar } f_2(x_1, x_2) = \frac{g(x_2)}{x_1} \tag{5.12}$$

donde:

$$g(x_2) = 2.0 - \exp\left\{-\left(\frac{x_2 - 0.2}{0.004}\right)^2\right\} -$$

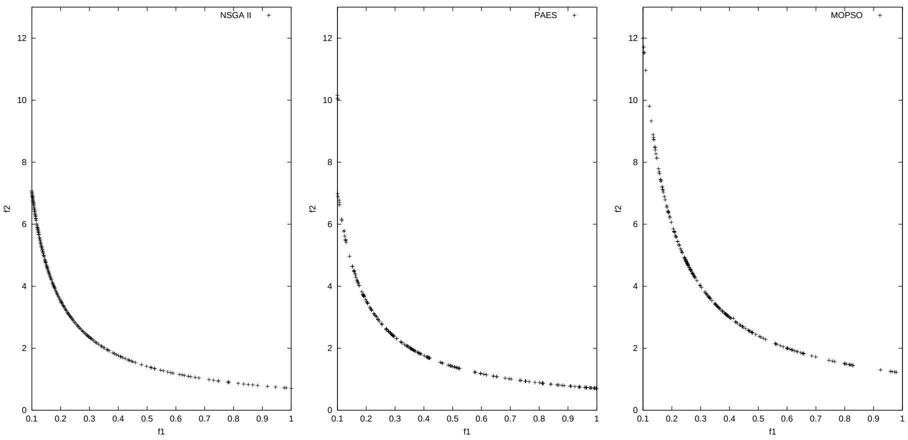

Figura 5.4: Frentes de Pareto producidos por el NSGA-II (izquierda), PAES (en medio), y MOPSO (derecha) para la cuarta función de prueba.

$$0.8 \exp \left\{ - \left(\frac{x_2 - 0.6}{0.4} \right)^2 \right\} \tag{5.13}$$

y $0.1 \leq x_1 \leq 1.0$, $0.1 \leq x_2 \leq 1.0$.

Las gráficas de la figura 5.4 muestran los frentes de Pareto producidos por el NSGA-II, PAES y MOPSO, respectivamente, para la cuarta función de prueba. Para este ejemplo, los tres algoritmos llevaron a cabo 12000 evaluaciones de la función objetivo. Los tiempos promedio de ejecución de cada algoritmo (20 corridas) fueron los siguientes: 6.864 segundos para el NSGA-II, 3.041 segundos para el PAES y sólo 0.8235 segundos para el MOPSO. Los valores promedio para la métrica M_1^* fueron los siguientes: 0.064788340 para el NSGA-II (con una desviación estándar de 0.014734278), 0.182493025 para el PAES (con una desviación estándar de 0.305695749), y 0.397989355 para el MOPSO (con una desviación estándar de 0.332828453).

5.2.5 Función de Prueba 5

El quinto ejemplo fue propuesto por Fonseca [15]:

$$\text{Minimizar } f_1(\vec{x}) = 1 - exp \left(- \sum_{i=1}^{n} \left(x_i - \frac{1}{\sqrt{n}} \right)^2 \right) \tag{5.14}$$

$$\text{Minimizar } f_2(\vec{x}) = 1 - exp \left(- \sum_{i=1}^{n} \left(x_i + \frac{1}{\sqrt{n}} \right)^2 \right) \tag{5.15}$$

donde: $-4 \leq x_i \leq 4$; $i = 1, 2, 3$.

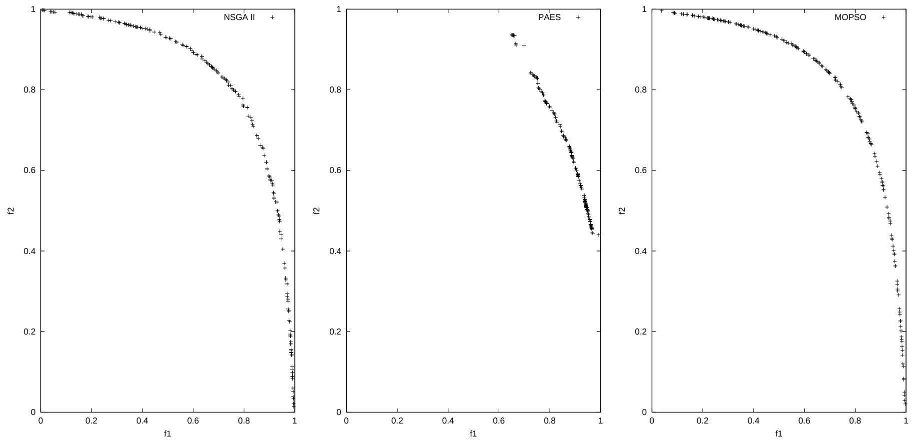

Figura 5.5: Frentes de Pareto producidos por el NSGA-II (izquierda), PAES (en medio), y MOPSO (derecha) para la quinta función de prueba.

Las gráficas de la figura 5.5 muestran los frentes de Pareto producidos por el NSGA-II, PAES y MOPSO, respectivamente, para la quinta función de prueba. Para este ejemplo, los tres algoritmos llevaron a cabo 3200 evaluaciones de la función objetivo. Los tiempos promedio de ejecución de cada algoritmo (20 corridas) fueron los siguientes: 2.3865 segundos para el NSGA-II, 4.0285 segundos para el PAES y sólo 0.133 segundos para el MOPSO. Los valores promedio para la métrica M_1^* fueron los siguientes: 0.133797950 para el NSGA-II (con una desviación estándar de 0.003387463), 0.185013150 para el PAES (con una desviación estándar de 0.021476946), y 0.138983350 para el MOPSO (con una desviación estándar de 0.003273670).

5.2.6 Función de Prueba 6

El sexto ejemplo fue propuesto por Poloni[29]:

$$\text{Maximizar } f_1(x_1, x_2) = -\left[1 + (A_1 - B_1)^2 + (A_2 - B_2)^2\right] \tag{5.16}$$

$$\text{Maximizar } f_2(x_1, x_2) = -\left[(x_1 + 3)^2 + (x_2 + 1)^2\right] \tag{5.17}$$

donde: $-3.1416 \leq x_1, x_2 \leq 3.1416$,

$$A_1 = 0.5 \sin 1 - 2 \cos 1 + \sin 2 - 1.5 \cos 2 \tag{5.18}$$

$$A_2 = 1.5 \sin 1 - \cos 1 + 2 \sin 2 - 0.5 \cos 2 \tag{5.19}$$

$$B_1 = 0.5 \sin x_1 - 2 \cos x_1 + \sin x_2 - 1.5 \cos x_2 \tag{5.20}$$

$$B_2 = 1.5 \sin x_1 - \cos x_1 + 2.0 \sin x_2 - 0.5 \cos x_2 \tag{5.21}$$

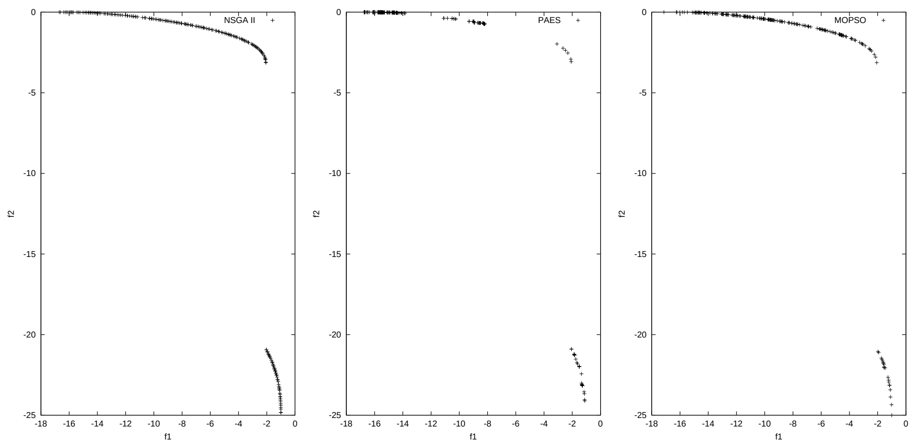

Figura 5.6: Frentes de Pareto producidos por el NSGA-II (izquierda), PAES (en medio), y MOPSO (derecha) para la quinta función de prueba.

Las gráficas de la figura 5.6 muestran los frentes de Pareto producidos por el NSGA-II, PAES y MOPSO, respectivamente, para la sexta función de prueba. Para este ejemplo, los tres algoritmos llevaron a cabo 4800 evaluaciones de la función objetivo. Los tiempos promedio de ejecución de cada algoritmo (20 corridas) fueron los siguientes: 3.727 segundos para el NSGA-II, 1.984 segundos para el PAES y sólo 0.169 segundos para el MOPSO. Los valores promedio para la métrica M_1^* fueron los siguientes: 0.017165200 para el NSGA-II (con una desviación estándar de 0.007175080), 0.064773000 para el PAES (con una desviación estándar de 0.156300632), y 0.029212510 para el MOPSO (con una desviación estándar de 0.013688876).

5.3 Discusión de Resultados

En todos los experimentos realizados[2], el MOPSO se comportó mejor que el NSGA-II y PAES en términos de la métrica usada (producía valores más bajos de M_1^* con una menor desviación estándar[3]) como en términos de la cantidad de tiempo (requería menos tiempo que las otras técnicas). Más aún, la representación gráfica de los frentes de Pareto producidos en cada caso también muestran que el MOPSO tiene una mejor dispersión a lo largo del frente del Pareto que los otros algoritmos y también estuvo más cerca a los frentes de Pareto deseados (ver figuras 5.1, 5.2, 5.3, 5.4, 5.5, 5.6).

[2]Exceptuando el experimento número 4, en donde el MOPSO fue ligeramente superado por las otras 2 técnicas.

[3]Excepto en los experimentos número 1, 2 y 6, en donde la desviación estándar del NSGA-II fue menor

Finalmente, queremos mencionar brevemente cuales son los parámetros usados por el MOPSO y algunas sugerencias de cómo definirlos:

- **Número de partículas**: Esto es equivalente al tamaño de la población de un algoritmo genético. Se recomienda usar entre 20 y 80 partículas. En esta tesis se usaron 80 partículas.
- **Número de ciclos**: Este parámetro está ligado con el número de partículas. La relación tiende a ser inversamente proporcional (es decir, a mayor número de partículas, menor número de ciclos y viceversa). Se recomienda usar entre 80 y 120.
- **Número de divisiones**: Esto nos permite determinar el número de hipercubos que serán generados en el espacio de la función objetivo. Se recomienda usar entre 30 y 50 divisiones. En esta tesis se usaron 30 divisiones en todos los experimentos.
- **Tamaño del repositorio**: Este parámetro es usado para delimitar el máximo número de vectores no dominados que pueden ser almacenados en el repositorio. El valor de este parámetro determina la calidad del frente de Pareto que se produzca. En los experimentos reportados en esta tesis se usó un valor de 200.

Conclusiones y trabajo futuro

En el capítulo 3, cuando hablamos sobre la extensión de la técnica PSO para manejar restricciones, se habló de la importancia que tenía el uso de coeficientes adecuados para que el desempeño de la técnica mejorara. A este respecto valdría la pena hacer un estudio estadístico y de sensibilidad, ya que en este trabajo, dados sus alcances y objetivos, no se tuvo el tiempo suficiente para realizarlo. Con este análisis se podría hacer notar la forma en que el algoritmo es afectado por los coeficientes W, $C1$ y $C2$ de la fórmula para calcular la velocidad, y mediante este estudio saber la forma en la que el algoritmo puede ser mejorado o simplemente saber qué parámetros pueden hacer que el algoritmo trabaje de una manera más eficiente.

En lo que respecta a la parte del MOPSO, es importante indicar que el PSO es una técnica de búsqueda sin restricciones. Es por eso, que es necesario el desarrollar un mecanismo adicional para trabajar con problemas de optimización multiobjetivo restringidos. En este trabajo sólo nos hemos enfocado a la generación de vectores no dominados y en el mecanismo para mantener diversidad, pero no hemos trabajado con funciones de prueba restringidas. La extensión para manejar problemas restringidos es, de cualquier forma, no muy difícil de desarrollar y la estamos llevando a cabo.

Un análisis de sensibilidad está también en desarrollo, para de esta forma poder determinar el papel de cada uno de los parámetros usados en el desempeño del algoritmo MOPSO.

Bibliografía

[1] Y. J. Cao and Q. H. Wu. Mechanical design optimization by mixed-variable evolutionary programming. In Zbigniew Michalewicz Thomas Bäck and Xin Yao, editors, *Proceedings of the 1997 International Conference on Evolutionary Computation*, pages 443–446, Indianapolis, Indiana, 1997. IEEE.

[2] Carlos A. Coello Coello. A Comprehensive Survey of Evolutionary-Based Multiobjective Optimization Techniques. *Knowledge and Information Systems. An International Journal*, 1(3):269–308, August 1999.

[3] Carlos A. Coello Coello. Functions used to test constraint-handling techniques. Technical report, LANIA-RI-2000-04, Laboratorio Nacional de Informática Avanzada, 2000.

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

[5] Carlos A. Coello Coello. Use of a self-adaptative penalty approach for engineering optimization problems. *Computer in Industry, 41(2):113-127*, January 2000.

[6] Nichal Lynn Cramer. A representation for the adaptative generation of simple sequential programs. In John J. Grefenstette, editor, *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, pages 183–187, Hillsdale, New Jersey, 1985. Lawrence Erlbaum Associates.

[7] K. A. De Jong. *An analysis of the behavior of a class of genetic adaptive systems.* PhD thesis, Dept. of Computer and Communication Sciences, University of Michigan, Ann Arbor, 1975.

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

[9] Kalyanmoy Deb. Optimal design of a welded beam via genetic algorithms. *AIAA Journal, 29(11):2013-2015*, November 1991.

[10] Kalyanmoy Deb. Geneas: A robust optimal design technique for mechanical component design. *In Dipankar Dasgupta and Zbigniew Michalewicz, editors, Evolutionary Algorithms in Engineering Applications*, pages 497–514, Springer-Verlag, Berlin, 1997.

[11] Kalyanmoy Deb, Samir Agrawal, Amrit Pratab, and T. Meyarivan. A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II. In *Proceedings of the Parallel Problem Solving from Nature VI Conference*, pages 849–858. Springer, 2000.

[12] Kalyanmoy Deb and David E. Goldberg. An Investigation of Niche and Species Formation in Genetic Function Optimization. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 42–50, San Mateo, California, June 1989. George Mason University, Morgan Kaufmann Publishers.

[13] Rusell C. Eberhart and Yuhui Shi. Particle swarm optimization: Developments, applications and resources. *Proc. Congress on Evolutionary Computation*, pages 81–86, 2001. Seoul, Korea. Piscataway, NJ: IEEE Service Center.

[14] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution.* Jonh Wiley and Sons, New York, 1966.

[15] 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.

[16] Carlos M. Fonseca and Peter J. Fleming. An Overview of Evolutionary Algorithms in Multiobjective Optimization. *Evolutionary Computation*, 3(1):1–16, Spring 1995.

[17] R. M. Friedberg. A learning machine: Part I. *IBM Journal of Research and Development*, 2:2–13, 1958.

[18] R. M. Friedberg, B. Dunham, and J. H. North. A learning machine: Part II. *IBM Journal of Research and Development*, 3:282–287, 1959.

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

[20] Jörg Heitkötter and David Beasley. The hitch-hiker's guide to evolutionary computation: A list of frequently asked questions (faq), eds. (2000). Available via anonymous FTP from ftp://rtfm.mit.edu/pub/usenet/news.answers/ai-faq/genetic/.

[21] Hisao Ishibuchi and Tadahiko Murata. Multi-Objective Genetic Local Search Algorithm. In Toshio Fukuda and Takeshi Furuhashi, editors, *Proceedings of the 1996 International Conference on Evolutionary Computation*, pages 119–124, Nagoya, Japan, 1996. IEEE.

[22] R. G. Fenton J. F. Fu and W. L. Cleghorn. A mixed integer-discrete continuos programming method and its applications to engineering design optimization. *Engineering Optimization, 17(3):263-280*, 1991.

[23] B.K. Kannan and S.N. Kramer. An augmented lagrange multiplier based method for mixed integer discrete continuous optimization and its applications to mechanical design. *Journal of Mechanical Design. Transactions of the ASME, 116:318-320.*, 1994.

[24] James Kennedy and Russell C. Eberhart. Particle swarm optimization. *Proceedings of the IEEE International Conference on Neural Networks*, pages 1942–1948, 1995. IEEE service Center.

[25] James Kennedy and Russell C. Eberhart. *Swarm Intelligence.* Morgan Kaufmann Publishers, San Francisco, California, 2001.

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

[27] John Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* The MIT Press, Cambridge, MA, 1992.

[28] Marco Laumanns, Eckart Zitzler, and Lothar Thiele. On the Effects of Archiving, Elitism, and Density Based Selection in Evolutionary Multi-objective 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 181–196. Springer-Verlag. Lecture Notes in Computer Science No. 1993, 2001.

[29] Carlo Poloni and Valentino Pediroda. GA coupled with computationally expensive simulations: tools to improve efficiency. In D. Quagliarella, J. Périaux, C. Poloni, and G. Winter, editors, *Genetic Algorithms and Evolution Strategies in Engineering and Computer Science. Recent Advances and Industrial Applications*, chapter 13, pages 267–288. John Wiley and Sons, West Sussex, England, 1997.

[30] K. M. Ragsdell and D. T. Phillips. Optimal design of a class of welded structures using geometric programming. *ASME Journal of Engineering for Industries, 98(3):1021-1025*, 1976. Series B.

[31] Ingo Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Frommann-Holzboog Verlag, Stuttgart, Germany, 1973.

[32] R. Rucker. *Seek!* New York: Four Walls Eight Windows, 1999.

[33] Thomas P. Runarsson and Xin Yao. Stochastic ranking for constrained evolutionary optimization. *IEEE Transactions on Evolutionary Computation*, 4(3):284–294, September 2000.

[34] E. Sandgren. Nonlinear integer and discrete programming in mechanical design. *Proceedings of the ASME Design Technology Conference, pages 95-105*, Kissimine, Florida, 1988.

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

[36] Yuhui Shi and Russell C. Eberhart. Parameter selection in particle swarm optimization. In *Evolutionary Programming VII: Proc.*, pages 591–600. New York: Springer-Verlag, 1998.

[37] James Siddall. *Analytical Design-Making in Engineering Design*. Prentice-Hall, 1972.

[38] David A. Van Veldhuizen and Gary B. Lamont. Multiobjective Evolutionary Algorithms: Analyzing the State-of-the-Art. *Evolutionary Computation*, 7(3):1–26, 2000.

[39] C. Zhang and H. P. Wang. Mixed-discrete nonlinear optimization with simulated annealing. *Engineering Optimization, 21:277-291*, 1993.

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