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

Manejo de restricciones usando optimización mediante cúmulos de partículas

Tesis que presenta

José Alfredo López Lara

Para obtener el grado de:

Maestro en Ciencias

En la especialidad de

Ingeniería Eléctrica
Opción Computación

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

México, D. F. Diciembre de 2005

Dedicatoria

A Julian y Yolanda.

Agradecimientos

A mis padres Julian y Yolanda que siempre están en mi corazón.

A mis hermanos Julio y Carmina por su apoyo incondicional.

A Bárbara por todo tu apoyo, cariño y amor que me diste.

A mis abuelos Rosendo y Nicolasa, Otilio y Eusebia por todo su amor que desde niño siempre recibí hasta hoy en día.

A mis tios que son seres queridos y que siempre me demuestran que juntos se pueden lograr muchos sueños.

Al Dr. Carlos A. Coello Coello que siempre que necesité de un buen consejo estuvo presente. También por brindarme la oportunidad de compartir su amistad y sus conocimientos.

A mis amigos que he conocido durante todo este tiempo Gregorio, Israel, Oscar, Rolando, Lorena, Luis Vicente, Mario, Mireya, Carlos, Jaime, Luis, Noel, Pancho, Gil, Jorge, Buenabad, Lucio, Isai, Omar, Sergio y esposa y a todas las personas que de alguna u otra forma formaron y forman parte de mi.

A Sofi por tus consejos, Flor y Feli por el apoyo y entusiasmo. Y el cariño que siempre recibí.

Agradezco el apoyo recibido por la biblioteca del departamento de Ingeniería Eléctrica del CINVESTAV muy en especial al personal administrativo.

Esta tesis se derivó del proyecto CONACYT titulado "Técnicas Avanzadas de optimización evolutiva multiobjetivo" (Ref. 45683-Y), cuyo responsable es el Dr. Carlos A. Coello Coello.

Al Consejo Nacional de Ciencia y Técnologia (CONACYT) por la beca otorgada durante dos años.

Y finalmente, no por eso menos importante, a Dios por darme la oportunidad de vivir este momento de mi vida.

Resumen

La optimización mediante cúmulos de partículas (*Particle Swarm Optimization* o *PSO*) es una heurística relativamente reciente que está inspirada en los patrones de vuelo de las aves [27]. Su éxito en los diversos problemas de optimización han hecho muy popular a esta heurística en una amplia variedad de disciplinas. Sin embargo, a pesar de esta popularidad, existen todavía muy pocas propuestas de mecanismos para manejo de restricciones que hayan sido diseñados específicamente para PSO.

Esta tesis precisamente introduce un nuevo esquema de manejo de restricciones diseñado especificamente para PSO, a la cual denominamos IPSO (*Improved Particle Swarm Optimization*). La propuesta es validada usando un conjunto estándar de funciones de prueba usado comunmente en la literatura especializada [32]. El desempeño del algoritmo propuesto se compara con respecto al de otras tres técnicas representativas del estado del arte en el área (jerarquías estocásticas [36], mapas homomorfos [29] y la propuesta por Toscano y Coello [34]).

El análisis de resultados indican que el mecanismo propuesto IPSO constituye una alternativa prometedora para el uso del PSO en optimización global con restricciones.

Abstract

Particle Swarm Optimization (PSO) is a relatively recent heuristic inspired on the flight patterns of a flock of birds [27]. Its success in diverse optimization problems have made PSO very popular in a wide variety of disciplines. However, despite its popularity, there still are very few proposals of constraint-handling mechanisms that have been specifically designed for PSO.

This masters thesis precisely introduces a new constraint-handling scheme which was specifically designed for PSO, and which is called IPSO (*Improved Particle Swarm Optimization*). The proposed approach is validated using a set of standard test functions that has been commonly adopted in the specialized literature [36]. The performance of the proposed approach is compared with respect to that of three other techniques which are representative of the state-of-the-art in the area (stochastic ranking [36], homomorphous maps [29] and the proposal of Toscano and Coello [34]).

The analysis of results indicates that the proposed IPSO constitutes a promising alternative for using PSO in constrained global optimization.

Índice general

Índice de figuras

Lista de Tablas

Lista de Algoritmos

Capítulo 1

Introducción

La "optimización mediante cúmulos de partículas" comúnmente conocida como PSO (por las siglas de su nombre en ingés: *Particle Swarm Optimization*) ha sido utilizada en diversas aplicaciones en los últimos años, gozando de gran popularidad y éxito. Esta técnica se inspiró en el vuelo de las parvadas de aves que salen en busca de comida [27]. PSO es comúnmente considerado un algoritmo evolutivo, al igual que los algoritmos genéticos, la programación evolutiva y las estrategias evolutivas.

En el campo de la optimización existen diferentes tipos de problemas, tales como la optimización global, optimización multiobjetivo y la optimización dinámica. Los algoritmos evolutivos, tales como el PSO, suelen utilizarse en optimización no lineal tanto mono-objetivo (global) como multiobjetivo. Sin embargo, a pesar de la popularidad del PSO como optimizador global, es muy escaso el trabajo existente en torno al diseño de esquemas para manejo de restricciones. Ese es precisamente el tema principal de esta tesis, en la cual se estudiará el uso de PSO en espacios de búsqueda restringidos.

Los intentos más importantes de adicionar mecanismos de restricciones en el PSO son los siguientes:

- Parsopoulos y Vrahatis propusieron en 2002 [33] un mecanismo de restricciones basado en la penalización multi-etapas no-estacionaria, es decir, que los valores de las penalizaciones se modifican dinámicamente. Este método no se validó con las funciones más representativas del área de manejo de restricciones.

- Hu y Eberhart [24], propusieron un esquema de penalización para problemas con restricciones no lineales, el cual consiste en crear partículas que no violen ninguna restricción. Los autores no indican cuál es el costo computacional para lograr que la población sea factible, aunque reportan pocas evaluaciones a la función de aptitud. En sus resultados

tampoco incluyen dos funciones de prueba (particularmente dificiles) que se encuentran en [36] (*g05 y g13*).

- Toscano Pulido y Coello Coello [34] propusieron un algoritmo para manejo de restricciones en PSO, el cual se aplica al momento de seleccionar un líder. Este esquema permite la selección de un líder aún cuando éste es infactible. Esto permite al PSO oscilar entre la región factible y no factible. Se reportan resultados competitivos, aunque, con menor robustez.

Sólo en el caso del mecanismo propuesto por Toscano Pulido y Coello Coello [34] se hacen comparaciones con la técnica más representativa del estado del arte propuesta por Tomas P. Runarsson y Xin Yao en 2000 [36]. En la presente tesis hacemos un análisis de nuestros resultados obtenidos y comparamos contra estas dos técnicas [34, 36] y una técnica adicional que son los mapas homomorfos propuesta por S. Koziel y Z. Michalewicz en el año de 1999 [29]. El número de evaluaciones a la función objetivo de nuestro algoritmo propuesto IPSO es de 350,000 a fin de poder hacer una comparación justa con los otros métodos [34, 36]. Cabe mencionar que en [29] el número de evaluaciones fue de 1,400,000.

Esta tesis está dividida en 6 capítulos y está organizada de la siquiente manera: En el capítulo 2 introducimos conceptos básicos de la computación evolutiva describiendo los diferentes paradigmas que la constituyen. Adicionalmente, vemos cómo el PSO tiene una estrecha relación con la computación evolutiva. En el capítulo 3 definimos lo que son problemas de optimización global con restricciones y revisamos los esquemas de manejo de restricciones más representativos del estado del arte, incluyendo los que se han propuesto para el PSO hasta la fecha. En el capítulo 4 describimos y justificamos el funcionamiento de nuestra técnica utilizada para resolver problemas de optimización global con restricciones (IPSO). En el capítulo 5, describimos las funciones de prueba adoptadas para nuestro estudio y los resultados obtenidos por nuestro algoritmo. En el último capítulo presentamos nuestras conclusiones y algunas posibles líneas de trabajo futuro.

Capítulo 2

Computación evolutiva

En el presente capítulo daremos una introducción a la computación evolutiva y sus principales paradigmas. Se incluye también una breve introducción a la técnica denominada *optimización mediante cúmulos de partículas* (PSO, por sus siglas en inglés), indicándose la relación que ésta guarda con la computación evolutiva.

2.1. Principios básicos de la computación evolutiva

La computación evolutiva (CE), se refiere a aquellas técnicas para resolver problemas que se basan en el principio de la selección natural, esto es, la supervivencia del más apto. La CE engloba a técnicas estocásticas de búsqueda y optimización. Su naturaleza estocástica implica que los resultados que producen estas técnicas varían de una ejecución a otra y de ahí que no se pueda garantizar que encontrarán siempre la mejor solución posible a un problema dado. Sin embargo, su desempeño promedio suele ser muy bueno en problemas de alta complejidad para los cuales los algoritmos deterministas requerirían costos computacionales prohibitivamente altos.

En términos generales, se puede decir que para poder simular el proceso evolutivo en una computadora se requiere:

- Codificar estructuras que se replicarán.
- Operaciones que afecten a los individuos.
- Una función de aptitud. RE
- Un mecanismo de selección.

El modelo en general de la computación evolutiva consiste de una *población* de individuos llamados *cromosomas* en los cuales se codifican las características de las soluciones potenciales a un problema. Estas características son los *genes* del individuo o del cromosoma y cada uno de sus valores se denomina *alelo*. En cada generación, los individuos compiten por su supervivencia, pero aquellos con las mejores capacidades tienen mayor oportunidad de reproducirse. La descendencia es generada mediante la combinación de partes de los cromosomas de los padres, es decir mediante la *cruza*. Adicionalmente, también suele presentarse algún tipo de *mutación* (o cambio aleatorio de los alelos de un cromosoma) que altera el cromosoma del individuo. La medida de las habilidades y capacidades del individuo se ve plasmada en su valor de *aptitud* que refleja cuál es el desempeño de la solución que representa dicho individuo para el problema propuesto [13].

2.2. Principales paradigmas

Hoy en día es cada vez más difícil distinguir las diferencias entre los distintos tipos de algoritmos evolutivos existentes. Sin embargo, por razones sobre todo históricas, suele hablarse de tres paradigmas principales [18, 12]:

1. Programación evolutiva.
2. Estrategias evolutivas.
3. Algoritmos genéticos.

Cada uno de estos paradigmas se originó de manera independiente y con motivaciones muy distintas. A continuación se presenta una revisión breve de cada uno de éstos.

2.2.1. Programación evolutiva

Lawrence J. Fogel propuso en los 1960s una técnica denominada "programación evolutiva", en la cual la inteligencia se ve como un comportamiento adaptativo [17, 18]. La programación evolutiva enfatiza los nexos de comportamiento entre padres e hijos, en vez de buscar emular operadores genéticos específicos (como en el caso de los algoritmos genéticos).

La programación evolutiva es una abstracción de la evolución a nivel de las especies, por lo que no se requiere el uso de un operador de recombinación (diferentes especies no se pueden cruzar entre sí). Asimismo, este algoritmo usa una selección probabilística.

Algunas aplicaciones de la programacion evolutiva son [16]:

Figura 2.1: Lawrence J. Fogel.

Algoritmo 1: Algoritmo de la programación evolutiva

```
1 begin
2     Generar aleatoriamente una población inicial.
3     Aplicar mutación.
4     Calcular la aptitud de cada hijo y usar un proceso de selección
      mediante torneo (normalmente estocástico) para determinar
      cuáles serán las soluciones que se retendrán.
5     Ir al paso 3 hasta cumplir la condición de paro.
6 end
```

- Predicción.
- Generalización.
- Juegos.
- Control automático.
- Problema del viajero.
- Planeación de rutas.
- Diseño y entrenamiento de redes neuronales.
- Reconocimiento de patrones.

2.2.2. Estrategias evolutivas

Las estrategias evolutivas (EEs) fueron desarrolladas en 1964 por un grupo de estudiantes de ingeniería de Alemania encabezados por Ingo Rechenberg, Hans-Paul Schwefel y Paul Bienert [2]. La motivación original de las EEs fue resolver problemas hidrodinámicos de alto grado de complejidad y que eran intratables con las técnicas clásicas de optimización de la época.

Las EEs son una abstracción a nivel de los individuos, por lo que sí existe un operador de cruza, que puede ser sexual (dos padres) o panmítica (más de

Figura 2.2: Ingo Rechenberg.

dos padres). Sin embargo, la cruza es un operador secundario y la mutación es el operador principal. Su selección es determinística y extintiva (los peores individuos de la población tienen cero probabilidades de sobrevivir).

La EE llamada $(\mu+1)-EE$ ya utiliza una población. Hay μ padres y se genera un solo hijo, el cual puede reemplazar al peor padre. Por otro lado, Schwefel introdujo en 1975 el uso de múltiples hijos en las EEs llamadas $(\mu+\lambda)-EE$ y $(\mu,\lambda)-EE$. En la primera, los mejores μ individuos obtenidos de la unión de padres e hijos se preservan (población traslapable). En la segunda, sólo los mejores μ hijos de la siguiente generación sobreviven (población no traslapable).

El procedimiento es el que utilizan las estrategias evolutivas (en su versión más general):

Algoritmo 2: Algoritmo básico de una estrategia evolutiva

1 **begin**
2 *Inicializar una población aleatoriamente.*
3 *Llevan a cabo la operación de recombinación usando μ padres para generar λ hijos.*
4 *Se efectúa la operación de mutación sobre los hijos.*
5 *Se evaluán ya sea sólo los hijos (si se trata de una estrategia (μ,λ)), o ambos, padres e hijos (si se trata de una estrategia $(\mu+\lambda)$).*
6 *Se selecciona a los μ individuos que formarán la nueva población.*
7 *Ir al paso 3 hasta cumplir la condición de paro.*
8 **end**

Las EEs se han aplicado a problemas como [37]:

- Ruteo y redes.
- Bioquímica.
- Óptica.

Figura 2.3: John H. Holland.

- Diseño de ingeniería.
- Magnetismo.

2.2.3. Algoritmos genéticos

En la década de los 1960s John H. Holland se interesó en estudiar los procesos involucrados en la adaptación. Inspirado por los estudios realizados en aquella época con autómatas celulares [5], y redes neuronales [38, 20], Holland se percató de que el uso de reglas simples podría generar comportamientos flexibles, y visualizó la posibilidad de estudiar la evolución de comportamientos en un mismo sistema complejo. Holland advirtió que un estudio de la adaptación debía reconocer que [20, 21]:

1. La adaptación ocurre en un ambiente.
2. La adaptación es un proceso poblacional.
3. Los comportamientos individuales pueden representarse mediante programas.
4. Pueden generarse nuevos comportamientos mediante variaciones aleatorias de los programas.
5. Las salidas de dos programas normalmente están relacionadas si sus estructuras están relacionadas.

De tal forma, Holland vio el proceso de adaptación en términos de un formalismo en el que los programas de una población interactúan y mejoran con base en un cierto ambiente que determina lo apropiado de su comportamiento. El combinar variaciones aleatorias con un proceso de selección (en función de qué tan apropiado fuese el comportamiento de un programa dado), debía entonces conducir a un sistema adaptativo general. En el año de 1975 Holland publicó un libro donde dio a conocer su sistema desarrollado a mediados de los 1960s, al cual denominó "plan reproductivo genético" [22], aunque después se popularizó bajo el nombre de "algoritmo genético". El

algoritmo genético (AG) trabaja a nivel del genotipo y su operador primario es la cruza sexual ya que modela la evolución a nivel de los individuos; el operador secundario de un AG es la mutación. La manera de seleccionar individuos es probabilística basada en aptitud.

Aunque concebido originalmente en el contexto del aprendizaje de máquina, el AG se ha utilizado mucho en optimización, siendo una técnica sumamente popular en la actualidad.

Algoritmo 3: Algoritmo básico de un algoritmo genéitco

```
1 begin
2     Inicializar una población aleatoriamente.
3     Calcular la aptitud de cada individuo.
4     Aplicar operadores de reproducción para producir la siguiente
      generación (cruza y mutación).
5     Ir al paso 3 hasta cumplir la condición de paro.
6 end
```

Algunas aplicaciones de los AGs se muestran a continuación [19]:

- Optimización (estructural, numérica, combinatoria).
- Aprendizaje de máquina (sistemas de clasificadores).
- Bases de datos (optimización de consultas).
- Reconocimiento de patrones (imágenes).
- Generación de gramáticas (regulares, libres de contexto).
- Planeación de movimientos de robots.
- Predicción.

2.3. Optimización mediante cúmulos de partículas

En 1995, James Kennedy y Russell Eberhart [26], propusieron una nueva heurística a la que denominaron "optimización mediante cúmulos de particulas" (*Particle Swarm Optimization* - PSO). La idea central de este paradigma es la de simular los movimientos de un grupo de aves o peces que intentan encontrar comida. La técnica puede ser vista como un algoritmo de comportamiento distribuido que lleva a cabo una búsqueda multidimensional. Son

Figura 2.4: James Kennedy.

muchas las similitudes que comparte la técnica del PSO con los algoritmos evolutivos, entre las cuales destacan las siguientes:

- Utilizan una población de posibles soluciones.
- Se calcula la aptitud de cada partícula de acuerdo a una función definida para el problema.
- Utilizan una fórmula para actualizar la velocidad de cada individuo (partícula) de manera análoga (aunque no equivalente) a un operador de mutación.

2.3.1. Algoritmo básico del PSO

El proceso original del PSO, tal y como lo muestran Eberhart y Shi en [11], se muestra en el siguiente algoritmo 4.
donde $VEL[i]$ es la velocidad de la partícula i, w es el peso inercial, R_1 y R_2 son valores aleatorios entre $[0, 1]$, C_1 y C_2 son valores definidos por el usuario que permanecen constantes a lo largo del proceso (normalmente su valor se encuentra entre $[0, 2]$), $PBEST[i]$ es el mejor valor (lugar) que la partícula i ha encontrado (visitado), $GBEST[i]$ es la mejor partícula de la población hasta la generación T, $POP[i]$ es el valor de la partícula i

A pesar de que el PSO tiene algunas diferencias con la CE (notablemente el uso de líderes para guiar la búsqueda), se le suele considerar un paradigma más de ella [26, 13].

2.3.2. Aplicaciones del PSO

Algunas aplicaciones del PSO son las siguientes [26, 14]:

- Balanceo de pesos en redes neuronales.
- Diseño de circuitos combinatorios.
- Optimización numérica.

Algoritmo 4: Algoritmo básico del PSO

1 **begin**

2 *Inicializar una población de partículas con posiciones y velocidades aleatorias en las* d *dimensiones dentro del espacio del problema.* ; *Evaluar el grado de aptitud con la función a optimizar con las* d *variables de decisión, para cada partícula.*

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 *Cambiar la velocidad y posición de la partícula de acuerdo a las ecuaciones siguientes:*

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

$$POP[i] = POP[i] + VEL[i] \tag{2.2}$$

5 *Ir al paso 3 hasta que se cumpla el criterio de detención. Dicho criterio de detención es un valor suficientemente bueno para la aptitud o un máximo número de iteraciones (generaciones).*

6 **end**

- Programación de tareas.
- Control óptimo.

Capítulo 3

Manejo de restricciones en computación evolutiva

Los ALGORITMOS de optimización en su forma original son técnicas para optimización sin restricciones. De ahí la necesidad de implementar técnicas que permitan incorporar la información pertinente sobre la violación de restricciones en la función de aptitud. En las siguientes secciones daremos una revisión a la definición del problema que es objeto de estudio en esta tesis y a las diferentes técnicas de penalización existentes. En cada caso, se discutirán las principales limitantes de cada técnica.

3.1. Definición del problema

Refiriéndonos al dominio de la optimización numérica, un problema a resolverse se describe de la siguiente manera:

Encontrar X tal que optimice $f(X)$

Sujeta a:

$$g_i(X) \leq 0, \qquad i = 1, ..., n \tag{3.1}$$

$$h_j(X) = 0, \qquad j = 1, ..., p \tag{3.2}$$

donde X es un vector de soluciones $X = [x_1, x_2, ..., x_r]^T$, n es el número de restricciones de desigualdad y p es el número de restricciones de igualdad. Ambos tipos de restricciones pueden ser tanto lineales como no lineales.

Denominaremos con κ a la zona factible del espacio de búsqueda β de tal forma que $(\kappa \subseteq \beta)$ como se muestra en la figura 3.1. Para una restricción de desigualdad que satisfaga $g_i(X) = 0$, entonces decimos que dicha restricción

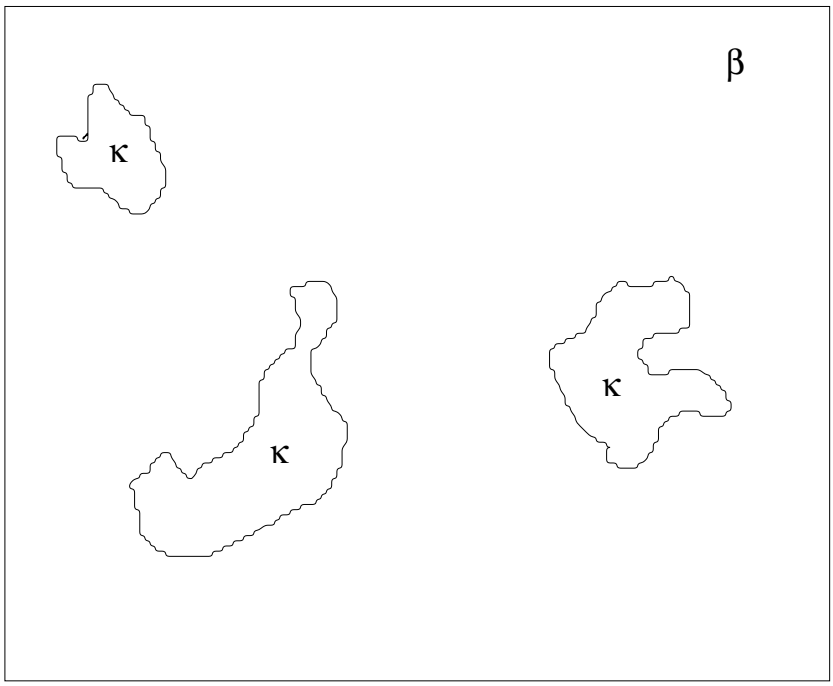

Figura 3.1: Espacio de búsqueda y región factible.

está activa en x. Todas las restricciones de igualdad h_j (sin importar del valor valor de x utilizado) las consideramos activas en todos los puntos de $\mathcal{F}$.

Existen problemas en los cuales la zona factible resulta ser pequeña y difícil de localizar dentro de un espacio de búsqueda muy grande (p. ej. cuando la zona factible es disjunta). Particularmente es en este tipo de problemas en los cuales resulta más útil el uso de heurísticas tales como el algoritmo genético. De tal forma, las técnicas evolutivas suelen utilizarse en optimización no lineal y con problemas que tienen funciones objetivo no diferenciables, con ruido o dinámicas. Asímismo, las técnicas evolutivas suelen utilizarse en problemas en los que tanto la forma del espacio de búsqueda como la ubicación exacta del óptimo global se desconocen.

3.2. Funciones de penalización

Son la técnica más común de incorporación de restricciones en la función de aptitud. La idea básica es extender el dominio de la función de aptitud usando:

$$fitness(X) = f_i(X) \pm Q_i \tag{3.3}$$

donde:

$$Q_i = c \times \sum_{(i=1)}^{n} g_i(X)^2 \tag{3.4}$$

para todas las restricciones violadas. En esta expresión, c es un factor de penalización definido por el usuario.

Hay al menos tres formas de penalizar a un individuo de acuerdo a su violación de las restricciones [35]:

- Puede penalizársele simplemente por no ser factible, sin usar ninguna información sobre qué tan cerca se encuentra de la región factible.
- Puede usarse la 'cantidad' de infactibilidad de un individuo para determinar su penalización correspondiente.
- Puede usarse el esfuerzo de 'reparar' al individuo (o sea, el costo de hacerlo factible) como parte de la penalización.

Richardson et al. [35] definieron algunas de las reglas básicas para diseñar una función de penalización:

1. Las penalizaciones que son funciones de la distancia a la zona factible son mejores que aquellas que son sólo funciones del número de restricciones violadas.
2. Para un problema con pocas variables y pocas restricciones, una penalización que sea sólo función del número de restricciones violadas no producirá ninguna solución factible.
3. Pueden construirse buenos factores de penalización a partir de 2 factores: el costo del cumplimiento máximo y el costo de cumplimiento esperado. El primero de ellos se refiere al costo de hacer factible a una solución infactible.
4. Las penalizaciones deben estar cerca del costo de cumplimiento esperado, pero no deben caer frecuentemente por debajo de él. Entre más preciso sea el factor de penalización, mejores resultarán las soluciones producidas. Cuando una penalización frecuentemente subestime el costo de cumplimiento, el proceso de búsqueda fallará.

Existen, sin embargo, varios problemas para definir una función de penalización:

1. No es obvio combinar los 2 factores de los que hablan Richardson et al. [35] en una función de penalización.

2. Definir los valores óptimos del factor de penalización es una tarea virtualmente imposible a menos que conozcamos el problema a fondo, en cuyo caso puede diseñarse una función de penalización a la medida, pero hacerlo resultará de cualquier forma innecesaria ya que el óptimo podrá determinarse por métodos analíticos exactos.

3. El costo del cumplimiento esperado normalmente tiene que estimarse mediante métodos alternativos (por ejemplo, estimando el grado de violación de las restricciones) que pueden ser difíciles de implementar.

A continuación revisaremos rápidamente diversas variantes de la función de penalización que se han propuesto en la literatura, comentando brevemente acerca de las ventajas y desventajas de cada una.

3.2.1. Pena de muerte

En las estrategias evolutivas ha sido popular una técnica a la que se le conoce como "pena de muerte", y que consiste en asignar una aptitud de cero a un individuo que no sea factible, y tomar el valor de la función de aptitud para los que sí lo sean [37, 3]. Otra variante de esta técnica consiste en desechar a los individuos infactibles, volviéndolos a generar nuevamente.

Análisis.

Esta técnica es muy eficiente porque no tenemos que recalcular las restricciones o la función objetivo ni tenemos que reparar a los individuos no factibles. Sin embargo, tiene varios inconvenientes. Por ejemplo, si la población inicial no tiene ningún individuo factible, la búsqueda no progresará (habrá estancamiento) porque todos los individuos tendrán la misma aptitud.

Algunos autores como Michalewicz [30] han explorado esta técnica, concluyendo que sólo puede usarse en espacios de búsqueda convexos y en aquellos casos en los que la zona factible constituya una parte razonablemente grande del tamaño total del espacio de búsqueda. Asimismo, esta técnica sólo puede lidiar con restricciones de desigualdad.

3.2.2. Penalizaciones estáticas

En esta técnica, introducida por Homaifar, Lai y Qi [23], la idea es definir varios niveles de violación y elegir un coeficiente de violación para cada uno de ellos, de manera que el coeficiente de penalización se incremente conforme alcanzamos niveles más altos de violación de las restricciones.

Un individuo se evalúa utilizando:

$$fitness(X) = f_i(X) + \sum_{j=1}^{n} R_{k,j} g_j \tag{3.5}$$

donde $R_{i,j}$ son los coeficientes de penalización utilizados y $k = 1, 2, ..., l$, siendo l los niveles de violación definidos por el usuario.

Análisis.

El principal problema de esta técnica es que requiere la definición de $m(2l + 1)$ parámetros, donde m se refiere al número de restricciones y l al número de niveles. De tal forma, que si tenemos, por ejemplo, un problema moderadamente pequeño, con 6 restricciones y 2 niveles, tendríamos que definir 30 parámetros, lo cual a todas luces es excesivo.

3.2.3. Penalizaciones dinámicas

Joines y Houck [25] propusieron una técnica en la que los factores de penalización cambian con respecto al tiempo.

Los individuos de la generación t se evalúan de acuerdo a:

$$fitness(X) = f_i(X) + (C \cdot t)^{\alpha} \sum_{j=1}^{n} |g_i(X)|^{\beta} \tag{3.6}$$

donde C, α, β son constantes definidas por el usuario. Los valores sugeridos por los autores para las constantes son: $C = 0{,}5$, $\alpha = 1$, $\beta = 1$ ó 2.

Análisis.

La técnica es muy susceptible a los valores de los parámetros y suele converger prematuramente cuando éstos no son seleccionados adecuadamente. Sin embargo, parece funcionar muy bien cuando la función objetivo es cuadrática.

3.2.4. Uso de recocido simulado

Michalewicz y Attia [31] consideraron una técnica para manejo de restricciones que usa el concepto de recocido simulado [28]: los coeficientes de penalización cambian cada cierto número de generaciones (cuando el algoritmo ha quedado atrapado en un óptimo local). Los individuos se evalúan usando:

$$fitness(X) = f_i(X) + \frac{1}{2\tau} \sum_{j=1}^{n} g_j(x)^2 \tag{3.7}$$

donde τ representa el horario de enfriamiento y es una función que debe ser definida por el usuario. Michalewicz y Attia sugieren usar: $\tau_0 = 1$ y

$\tau_f = 1 \times 10^{-6}$, con incrementos τ_{i+1} =$0.1\tau_i$. *Análisis* El principal problema de esta técnica es la definición del horario de enfriamiento.

3.2.5. Penalizaciones adaptativas

Bean y Hadj-Alouane [4] desarrollaron una técnica para adaptar penalizaciones con base en un proceso de retroalimentación del ambiente durante la ejecución de un algoritmo genético.

Cada individuo es evaluado utilizando:

$$fitness_i(X) = f_i(X) + \lambda(t) \sum_{j=1}^{n} g_i(X)^2 \tag{3.8}$$

donde $\lambda(t)$ se actualiza cada t generaciones usando las siguientes reglas:

$$\lambda(t) = \begin{cases} (1/\beta_1)\lambda(t) & \text{si caso \#1} \\ \beta_2\lambda(t) & \text{si caso \#2} \\ \lambda & \text{de lo contrario} \end{cases} \tag{3.9}$$

donde $\beta_1, \beta_2 > 1$ y toman valores diferentes (para evitar ciclos).

El caso #1 ocurre cuando el mejor individuo en las últimas k generaciones fue siempre factible. El caso #2 ocurre cuando dicho individuo no fue nunca factible. Esta técnica lo que hace entonces es disminuir la penalización cuando el mejor individuo resulta consistentemente factible y aumentar la penalización cuando resulta infactible. Si estos cambios son intermitentes (es decir, el mejor individuo es a veces factible y a veces no), entonces la penalización no se cambia. Obviamente, es necesario que el usuario defina el valor inicial λ_0.

Análisis.

El principal problema de esta técnica es cómo elegir el factor inicial de penalización y el intervalo generacional (o sea, k) de manera que el monitoreo de factibilidad produzca resultados razonables.

Una k muy grande o muy pequeña puede hacer que la función de penalización nunca cambie, en cuyo caso la técnica se reduce a una penalización estática tradicional.

Finalmente, tampoco está claro cómo definir buenos valores para β_1 y β_2.

3.2.6. Algoritmo genético segregado

Esta técnica fue propuesta por Le Riche et al. [15] y consiste en usar 2 funciones de penalización en vez de una, empleando para ello dos poblaciones. Estas dos funciones intentan balancear las penalizaciones moderadas con las fuertes. Inicialmente, se divide la población en dos grupos, de manera que los

individuos de cada grupo se evalúen usando un factor de penalización distinto. Después se elige a los mejores individuos de cada grupo para ser padres de la siguiente generación, lo que hace que se combinen individuos factibles con infactibles (se utilizan dos factores diferentes, uno grande y otro pequeño), manteniendo la diversidad y evitando quedar atrapado en 'optimos locales. En la implementación original de esta técnica se utilizaron jerarquías lineales para decrementar la presión de selección y la técnica se aplicó a un problema de diseño de elementos estructurales compuestos con gran éxito.

Análisis.

El principal inconveniente de esta técnica es la forma de elegir los factores de penalización de las dos poblaciones, lo cual es difícil de hacer cuando no hay información preliminar disponible sobre el problema que tratamos de resolver.

3.2.7. Penalización con base en la factibilidad

Esta técnica fue propuesta por Deb [9] y consiste en evaluar un individuo de acuerdo a:

$$fitness(X) = \begin{cases} f_i(X) & \text{Si la solución es factible} \\ f_{peor} + \sum_{j=1}^{n} g_j(X) & \text{de lo contrario} \end{cases} \tag{3.10}$$

donde f_{peor} es el valor de la función objetivo de la peor solución factible de la población. Si no hay ninguna solución factible en la población, entonces f_{peor} se hace igual a cero. Deb [9] usa torneo binario aplicando las siguientes reglas:

1. Una solución factible siempre es preferida sobre una no factible.

2. Entre dos soluciones factibles, la que tenga mejor valor de su función objetivo es seleccionada.

3. Entre dos soluciones no factibles, la que viole el menor número de restricciones es elegida.

Análisis.

Uno de los problemas de esta técnica es que tiene dificultades para mantener diversidad en la población, y para ello se requiere del uso de nichos [10] y porcentajes de mutación inusualmente altos.

3.2.8. Penalizaciones co-evolutivas

La idea de que el mismo proceso evolucione los factores de penalización fue utilizada por Coello [6, 7], quien propone el uso de dos subpoblaciones. En una evolucionarán las soluciones codificadas del problema y en la otra evolucionarán los factores de penalización los cuales multiplicarán a los siguientes factores: *coef*, que es la suma de los valores con los que las restricciones han sido violadas por un individuo y *viol*, el cual almacena el número de restricciones que han sido violadas por ese mismo individuo. La aptitud de una solución se calcula con:

$$fitness(X) = f(X) - (coef \cdot w_1 + viol \cdot w_2)$$

Análisis.

Las subpoblaciones implican la definición de parámetros extra como lo son el número de generaciones y el número de individuos para cada una de las dos subpoblaciones adoptadas. Por otro lado, la implementación es más laboriosa ya que se usan realmente dos AG's ejecutándose de manera alterna. Por último, este enfoque sólo trabaja con restricciones de desigualdad.

3.2.9. Mapas homomorfos

Propuesta en 1999 por Koziel y Michalewicz [29], esta técnica realiza un mapeo de la zona factible con un cubo n-dimensional. Se propusieron dos variantes de esta técnica: una relativamente simple para zonas factibles convexas y otra mucho más elaborada, para zonas factibles no convexas. Ambas son utilizadas conjuntamente con un algoritmo genético con decodificación binaria con códigos de Gray.

A continuación veremos las características de los dos casos manejados por esta técnica.

- Zona factible convexa.
  - Se define un punto arbitrario factible que define un segmento rectilíneo hacia la frontera de la zona factible.
  - El buen desempeño del algoritmo depende de la localización del punto arbitrario para que el mapeo sea adecuado.
  - La ubicación utópica de este punto sería en el centro geométrico de la zona factible.
- Zona factible no convexa.

- En este caso el segmento definido por el punto arbitrario factible puede intersectar en varias ocasiones los límites de la zona factible del espacio de búsqueda.
- Por lo tanto, se agrega un mapeo adicional del espacio de búsqeda definido por los intervalos de las variables de decisión.
- Después se obtienen las partes factibles del segmento rectilíneo.
- Luego se unen estos dos segmentos mediante otro mapeo. Un parámetro u definido por el usuario es necesario para este caso.

Análisis.

Esta técnica requiere de un parámetro adicional u para el proceso de mapeo. La calidad de sus resultados reportados es muy buena pero el número de evaluaciones requerido es grande en comparación con otras propuestas (1,400,000). Adicionalmente, este metodo es complicado de implementar, y el costo computacional de cada evaluación realizada es alto para el caso no convexo, ya que se debe realizar un proceso iterativo para localizar la frontera entre la zona factible y la no factible.

3.2.10. Jerarquías estocásticas

La técnica de jerarquías Estocasticas fue propuesta por Runarsson y Yao [36]. Esta técnica intenta balancear la influencia de la función objetivo y una función de penalización en la asignación de aptitud de un individuo. El método utiliza una selección basada en jerarquías. Esta técnica no requiere la definición de un factor de penalización, pero requiere de un parámetro P_f que determina el balance entre la función objetivo y la función de penalización. Las soluciones se ordenan utilizando un algoritmo tipo "bubble-sort" (burbuja). Dependiendo del valor de P_f, la comparación entre 2 individuos adyacentes se realiza con base sólo en el valor de la función objetivo. Las comparaciones restantes se hacen con base en la suma de violación de restricciones (en otras palabras, se usa una penalización estática).

Análisis.

Las jerarquías estocásticas utilizan una estrategia evolutiva como motor de búsqueda. La técnica es sensible al parámetro P_f aunque sus autores realizaron un estudio a partir del cual se sugieren los valores más adecuados de este parámetro. El número de evaluaciones de la función objetivo es menor a las técnicas anteriores (350,000) y el metodo es sencillo de implementar. Adicionalmente, este método es uno de los mejores disponibles hoy en día, por lo cual se usa frecuentemente como la técnica contra la cual comparar nuevos métodos de manejo manejo de restricciones usados con algoritmos evolutivos.

Algoritmo 5: Algoritmo de las jerarquías estocásticas propuesto en [36].

```
for i = 0 to N do
  for j = 0 to P − 1 do
    u = random(0,1)
    if φ(I_j) = φ(I_{j+1}) = 0 OR (u < P_f) then
      if f(I_j) > f(I_{j+1}) then
        Intercambiar (I_j, I_{j+1})
      end if
    else if (φ(I_j) > φ(I_{j+1}) then
      Intercambiar (I_j, I_{j+1} )
    end if
  end for
end for
```

3.3. Métodos de manejo de restricciones basados en PSO

En los últimos años, la optimización mediante cúmulos de partículas (PSO) ha incrementado su popularidad, principalmente en las tareas de optimización numérica [35]. Sin embargo, el PSO, al igual que los demás algoritmos evolutivos, carece de un mecanismo explícito para incorporar restricciones. De tal forma, el desarrollo de mecanismos para manejo de restricciones en PSO es un área abierta de investigación. A continuación presentaremos los trabajos más representativos sobre manejo de restricciones en PSO.

3.3.1. Método basado en PSO para manejo de restricciones

Parsopoulos y Vrahatis propusieron en 2002 [33] el uso de una función de penalización multi-etapas no-estacionaria, es decir, que los valores de las penalizaciones se modifican dinámicamente con respecto al número de iteración actual.

Análisis.

Los autores experimentaron con tres variantes de PSO y comparan sus resultados con respecto a una estrategia evolutiva usando seis funciones de prueba. Los autores realizaron 100000 evaluaciones de la funcion de aptitud, pero no usaron la mayoría de las funciones de prueba estándar reportadas en la literatura especializada para manejo de restricciones [36] .

3.3.2. Optimización de problemas no lineales con PSO

Hu y Eberhart [24], propusieron un esquema de penalización para problemas con restricciones no lineales, el cual consiste en crear partículas que no violen ninguna restricción. Con esto se consigue tener una población en la región factible del problema. Posteriormente tratan de mejorar esas soluciones y encontrar el óptimo de la función.

Análisis.

Las funciones de prueba para este esquema fueron doce de las trece funciones reportadas en [36]. Aunque en general se reportan pocas evaluaciones (entre 5,000 y 20,000), los autores no indican cuál es el costo computacional (adicional) requerido para alcanzar la zona factible. Este es un punto muy importante ya que en algunos problemas de prueba la zona factible es muy pequeña y el costo de alcanzarla puede ser muy alto. También se hace notar que los autores realizaron modificaciones tanto en el número de iteraciones como en el tamaño de la población y hasta en el modelo de PSO adoptado (local o global), lo que es indicativo de un esquema poco robusto y poco generalizable.

3.3.3. Mecanismo para manejo de restricciones basado en PSO

Toscano Pulido y Coello Coello [34] propusieron un algoritmo para manejo de restricciones en PSO, el cual se aplica al momento de seleccionar un líder. El enfoque se basa en el uso de reglas de comparación muy simples:

1. Al comparar dos partículas factibles, la que tenga la aptitud más alta gana.
2. Si se compara una partícula factible con una infactible, la factible gana.
3. Si las dos partículas que se comparan son infactibles, entonces la que tenga la menor cantidad total de violación de las restricciones es la que gana.

Análisis.

En este esquema, la selección del líder da preferencia a la partícula que se encuentre más cerca de la región factible, aun cuando ésta sea infactible. Esto permite al PSO oscilar entre la zona factible y la no factible. Esta técnica fue validada con las trece funciones reportadas en [36], obteniendo buenos resultados aunque con menor robustez que el método de jerarquías estocásticas.

3.4. Conclusiones de los métodos de manejo de resticciones para PSO

Tras analizar los tres métodos de manejo de restricciones basadas en PSO previamente descritos, resulta evidente la necesidad de desarrollar más trabajo en esta área. Claramente, los métodos de Parsopoulos y Vrahatus, así como el de Hu y Eberhart no fuerón validados adecuadamente y podrían no resultar adecuados para problemas en los que el espacio de búsqueda está altamente restringido. La propuesta que luce hasta ahora como más adecuada es la de Toscano Pulido y Coello Coello, por lo que en esta tesis hacemos una nueva propuesta encaminada precisamente a mejorar su desempeño.

3.5. Funciones de prueba para manejo de restricciones

Con el objeto de probar la eficiencia del algoritmo propuesto en esta tesis. Se adoptó el conjunto estándar de funciones de prueba disponibles en la literatura para probar técnicas de manejo de restricciones [29, 36]. Todas estas funciones se refieren a problemas de optimización numérica global previamente resueltas con técnicas evolutivas y con técnicas clásicas. Este conjunto de funciones de prueba incluye porblemas con restricciones de desigualdad y de igualdad, espacios de búsqueda de alta y baja dimensionalidad, espacios convexos y no convexos, además de zonas factibles disjuntas y/o muy pequeñas.
En la tabla 3.1, se resumen las características de las trece funciones de prueba adoptadas para evaluar el algoritmo propuesto. LI indica el número de desigualdades lineales, NI indica el número de desigualdades no lineales, LE indica el número de igualdades lineales y NE el número de igualdades no lineales, n indica el número de variables de decisión. ρ representa una estimación del tamaño de la región factible con respecto a todo el espacio de búsqueda. ρ se obtiene generando un millón de soluciones aleatorias, y evaluando qué porcentaje de ellas son factibles.

Puede observarse que hay funciones que tienen la zona factible muy pequeña como en el caso de g05, g07 y g13 en las que $\rho = 0{,}0000\,\%$, es decir, no se encuentra ningún individuo factible de manera aleatoria, lo que implica que en esos casos resultará desafiante el llegar siquiera a la zona factible.

Existen otros casos como en g02 donde la zona factible es muy grande pero llegar al óptimo es muy complicado ya que esta función tiene muchos óptimos locales. g02, g03 y g12 son problemas de maximización y los casos restantes son problemas de minimización. Sin embargo, para facilitar la presentación

Problema	n	Función	ρ	LI	NI	LE	NE
g01	13	cuadrática	0,0003 %	9	0	0	0
g02	20	no lineal	99,9973 %	1	1	0	0
g03	10	no lineal	0,0026 %	0	0	0	1
g04	5	cuadrática	27,0079 %	0	6	0	0
g05	4	no lineal	0,0000 %	2	0	0	3
g06	2	no lineal	0,0057 %	0	2	0	0
g07	10	cuadrática	0,0000 %	3	5	0	0
g08	2	no lineal	0,8581 %	0	2	0	0
g09	7	no lineal	0,5199 %	0	4	0	0
g10	8	lineal	0,0020 %	3	3	0	0
g11	2	cuadrática	0,0973 %	0	0	0	1
g12	3	cuadrática	4,7697 %	0	9^3	0	0
g13	5	no lineal	0,0000 %	0	0	1	2

Tabla 3.1: Valores de ρ de las trece funciones de prueba.

de resultados se manejarán todos los problemas como de minimización.

Estas funciones fueron propuestas originalmente en [32] y extendidas en [36]. Estas trece funciones contienen características que las hacen difíciles de resolver para cualquier tipo de técnica (incluyendo algoritmos evolutivos), tales como zonas factibles disjuntas, igualdades no lineales, multimodalidad, etc.

3.5.1. Problema g01

A. g01
Minimizar

$$f(x) = 5\sum_{i=1}^{4} X_i - 5\sum_{i=1}^{4} x_i^2 - \sum_{i=5}^{13} x_i \tag{3.11}$$

sujeto a:

$$g_1(x) = 2x_1 + 2x_2 + x_{10} + x_{11} - 10 \leq 0$$

$$g_2(x) = 2x_1 + 2x_2 + x_{10} + x_{12} - 10 \leq 0$$

$$g_3(x) = 2x_2 + 2x_3 + x_{11} + x_{12} - 10 \leq 0$$

$$g_4(x) = -8x_1 + x_{10} \leq 0$$

$$g_5(x) = -8x_2 + x_{11} \leq 0$$

$$g_6(x) = -8x_3 + x_{12} \leq 0$$

$$g_7(x) = -2x_4 - x_5 + x_10 \leq 0$$

$$g_8(x) = -2x_6 - x_7 + x_11 \leq 0$$

$$g_9(x) = -2x_8 - x_9 + x_12 \leq 0$$

donde los rangos son $0 \leq x_i \leq 1$ $(i = 1, ..., 9)$, $0 \leq x_i \leq 100$ (i= 10, 11, 12) y $0 \leq x_{13} \leq 1$. El mínimo global esta en $x^* = (1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 1)$ donde 6 restricciones están activas (*g1, g2, g3, g7, g8* y *g9*) y $f(x^*) = 15$.

3.5.2. Problema g02

B. g02
Maximizar

$$f(x) = \frac{\sum_{i=1}^{n} cos^4(x_i) - 2\prod_{i=1}^{n} cos^2(x_i)}{\sqrt{\sum_{i=1}^{n} ix_i^2}} \tag{3.12}$$

sujeto a:

$$g_1(x) = 0{,}75 - \prod_{i=1}^{n} x_i \leq 0$$

$$g_2(x) = \sum_{i=1}^{n} x_i - 7{,}5n \leq 0$$

donde $n = 20$ y $0 < x_i < 10$ (i =1,...,n). El máximo global es desconocido y la mejor solución reportada en la literatura especializada es [36] $f(x^*) =$ 0,803619. La restricción *g1* está casi activa ($g_1 = -10^{-8}$).

3.5.3. Problema g03

C. g03
Minimizar

$$f(x) = -(\sqrt{n})^n \prod_{i=1}^{n} x_i \tag{3.13}$$

sujeto a:

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

donde $n = 10$ y $0 \leq x_i \leq 1$ (i =1,...,n). El máximo global está en $x_i^* = 1/\sqrt{n}(i$ =1,...,$n)$ donde $f(x^*) = 1$.

3.5.4. Problema g04

D. g04
Minimizar

$$f(x) = 5{,}3578547x_3^2 + 0{,}8356891x_1x_2 + 37{,}293239x_1 - 40792{,}141 \quad (3.14)$$

sujeto a:

$$g_1(x) = 85{,}334407 + 0{,}0056858x_2x_5 + 0{,}0006262x_1x_4 - 0{,}0022053x_3x_5 - 92 \leq 0$$

$$g_2(x) = -85{,}334407 - 0{,}0056858x_2x_5 - 0{,}0006262x_1x_4 + 0{,}0022053x_3x_5 \leq 0$$

$$g_3(x) = 80{,}51249 + 0{,}0071317x_2x_5 + 0{,}0029955x_1x_2 + 0{,}0021813x_3^2 - 110 \leq 0$$

$$g_4(x) = -80{,}51249 - 0{,}0071317x_2x_5 - 0{,}0029955x_1x_2 - 0{,}0021813x_3^2 + 90 \leq 0$$

$$g_5(x) = 9{,}300961 + 0{,}0047026x_3x_5 + 0{,}0012547x_1x_3 + 0{,}0019085x_3x_4 - 25 \leq 0$$

$$g_6(x) = -9{,}300961 - 0{,}0047026x_3x_5 - 0{,}0012547x_1x_3 - 0{,}0019085x_3x_4 + 20 \leq 0$$

donde $78 \leq x_1 \leq 102$, $33 \leq x_2 \leq 45$ y $27 \leq x_i \leq 45$ (i =3, 4, 5). El óptimo global está en $x^* = (78, 33, 29{,}995256025682, 45, 36{,}775812905788,)$ donde $f(x^*) = -30665{,}539$. Las restricciones *g1 y g6* están activas.

3.5.5. Problema g05

E. g05
Minimizar

$$f(x) = 3x_1 + 0{,}000001x_1^3 + 2x_2 + (0{,}000002/3)x_2^3 \quad (3.15)$$

sujeto a:

$$g_1(x) = -x_4 + x_3 - 0{,}55 \leq 0$$

$$g_2(x) = -x_3 + x_4 - 0{,}55 \leq 0$$

$$h_3(x) = 1000sin(x_3 - 0{,}25) + 1000sin(-x_4 - 0{,}25) + 894{,}8 - x_1 = 0$$

$$h_4(x) = 1000sin(x_3 - 0{,}25) + 1000sin(x_3 - x_4 - 0{,}25) + 894{,}8 - x_2 = 0$$

$$h_5(x) = 1000sin(x_4 - 0{,}25) + 1000sin(x_4 - x_3 - 0{,}25) + 1294{,}8 = 0$$

donde $0 \leq x_1 \leq 1200$, $0 \leq x_2 \leq 1200$, $-0{,}55 \leq x_3 \leq 0{,}55$ y $-0{,}55 \leq x_4 \leq 0{,}55$. La mejor solución conocida es [29] $x^* = (679.9453, 1026.067, 0.1188764, -0.3962336)$ donde $f(x^*) = 5126.4981$.

3.5.6. Problema g06

F. g06
Minimizar

$$f(x) = (x_1 - 10)^3 + (x2 - 20)^3 \tag{3.16}$$

sujeto a:

$$g_1(x) = -(x_1 - 5)^2 - (x_2 - 5)^2 + 100 \leq 0$$

$$g_2(x) = (x_1 - 6)^2 + (x_2 - 5)^2 - 82{,}81 \leq 0$$

donde $13 \leq x_1 \leq 100$ y $0 \leq x_2 \leq 100$. La mejor solución reportada para este problema es $x^* = (14{,}095, 0{,}84296)$ donde $f(x) = -6961{,}81388$. Ambas restricciones están activas, en el óptimo.

3.5.7. Problema g07

G. g07
Minimizar

$$\begin{aligned} f(x) \quad = \quad & x_1^2 + x_2^2 + x_1 x_2 - 14x_1 - 16x_2 + (x_3 - 10)^2 \\ & +4(x_4 - 5)^2 + (x_5 - 3)^2 \\ & +2(x_6 - 1)^2 + 5x_7^2 + 7(x_8 - 11)^2 + 2(x_9 - 10)^2 \\ & +(x_{10} - 7)^2 + 45 \end{aligned} \tag{3.17}$$

sujeto a:

$$g_1(\vec{x}) = -105 + 4x_1 + 5x_2 - 3x_7 + 9x_8 \leq 0$$

$$g_2(\vec{x}) = 10x_1 - 8x_2 - 17x_7 + 2x_8 \leq 0$$

$$g_3(\vec{x}) = -8x_1 + 2x_2 + 5x_9 - 2x_{10} - 12 \leq 0$$

$$g_4(\vec{x}) = 3(x_1 - 2)^2 + 4(x_2 - 3)^2 + 2x_3^2 - 7x_4 - 120 \leq 0$$

$$g_5(\vec{x}) = 5x_1^2 + 8x_2 + (x_3 - 6)^2 - 2x_4 - 40 \leq 0$$

$$g_6(\vec{x}) = x_1^2 + 2(x_2 - 2)^2 - 2x_1 x_2 + 14x_5 - 6x_6 \leq 0$$

$$g_7(\vec{x}) = 0{,}5(x_1 - 8)^2 + 2(x_2 - 4)^2 + 3x_5^2 - x_6 - 30 \leq 0$$

$$g_8(\vec{x}) = -3x_1 + 6x_2 + 12(x_9 - 8)^2 - 7x_{10} \leq 0$$

donde $-10 \leq x_i \leq 10$ $(i = 1, \ldots, 10)$. El óptimo global es $x^* = (2{,}171996,$ $2{,}363683, 8{,}773926, 5{,}095984, 0{,}9906548, 1{,}430574, 1{,}321644, 9{,}828726,$ $8{,}280092, 8{,}375927)$ donde $f(x^*) = 24{,}3062091$. Las restricciones g_1, g_2, g_3, g_4, g_5 y g_6 están activas.

3.5.8. Problema g08

H. g08

Minimizar

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

sujeto a:

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

donde $0 \leq x_1 \leq 10$ y $0 \leq x_2 \leq 10$. La mejor solución reportada hasta la fecha está localizada en $x^* = (1{,}2279713, 4{,}2453733)$ donde $f(x^*) = 0{,}095825$.

3.5.9. Problema g09

I. g09

Minimizar

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

sujeto a:

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

donde $-10 \leq x_i \leq 10$ $(i = 1, \ldots, 7)$. El óptimo global está en $x^* = (2{,}330499, 1{,}951372, -0{,}4775414, 4{,}365726, -0{,}6244870, 1{,}038131, 1{,}594227)$ donde $f(x^*) = 680{,}6300573$. Ambas restricciones están activas (g_1 y g_4).

3.5.10. Problema g10

J. g10
Minimizar

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

sujeto a:

$$g_1(\vec{x}) = -1 + 0{,}0025(x_4 + x_6) \leq 0$$

$$g_2(\vec{x}) = -1 + 0{,}0025(x_5 + x_7 - x_4) \leq 0$$

$$g_3(\vec{x}) = -1 + 0{,}01(x_8 - x_5) \leq 0$$

$$g_4(\vec{x}) = -x_1x_6 + 833{,}33252x_4 + 100x_1 - 83333{,}333 \leq 0$$

$$g_5(\vec{x}) = -x_2x_7 + 1250x_5 + x_2x_4 - 1250x_4 \leq 0$$

$$g_6(\vec{x}) = -x_3x_8 + 1250000 + x_3x_5 - 2500x_5 \leq 0$$

donde $100 \leq x_1 \leq 10000$, $1000 \leq x_i \leq 10000$, $(i = 2, 3)$, $10 \leq x_i \leq 1000$, $(i = 4, \ldots, 8)$. El óptimo global está en: $x^* = (579{,}19, 1360{,}13, 5109{,}92, 182{,}0174, 295{,}5985, 217{,}9799, 286{,}40, 395{,}5979)$, donde $f(x^*) = 7049{,}25$. g_1, g_2 y g_3 están activas.

3.5.11. Problema g11

K. g11
Minimizar

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

sujeto a:

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

donde $-1 \leq x_1 \leq 1$, $-1 \leq x_2 \leq 1$. La solución óptima es $x^* = (\pm 1/\sqrt{2}, 1/2)$ donde $f(x^*) = 0{,}75$.

3.5.12. Problema g12

L. g12
Minimizar

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

sujeto a:

$$g_1(\vec{x}) = (x_1 - p)^2 + (x_2 - q)^2 + (x_3 - r)^2 - 0{,}0625 \leq 0$$

donde $0 \leq x_i \leq 10$ $(i = 1, 2, 3)$ y $p, q, r = 1, 2, ..., 9$. El óptimo global está localizado en $x^* = (5, 5, 5)$ donde $f(x^*) = 1$.

3.5.13. Problema g13

M. g13
Minimizar

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

sujeto a:

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

donde $-2{,}3 \leq x_i \leq 2{,}3$ $(i = 1, 2)$ y $-3{,}2 \leq x_i \leq 3{,}2$ $(i = 3, 4, 5)$. La solución óptima es $x^* = (-1{,}717143, 1{,}595709, 1{,}827247, -0{,}7636413, -0{,}763645)$ donde $f(x^*) = 0{,}0539498$.

Capítulo 4

Descripción del algortimo IPSO

Dentro de este capítulo se describirán las extensiones propuestas al algoritmo básico del PSO a fin de poder resolver eficazmente problemas con restricciones.

Este nuevo esquema que se propone selecciona una topología (anillo *lbest* o completamente conectada *gbest*) y usa como operador principal la función de vuelo del propio PSO. La técnica para manejar restricciones que se adoptó es una extensión de la presentada en [34]. Finalmente, profundizaremos los aspectos más importantes del algoritmo propuesto con respecto a la versión original del PSO.

4.1. Algoritmo básico IPSO

Como se dijo en el Capítulo 2, la idea central de este paradigma es la de simular movimientos dentro de un grupo (sus creadores se inspiraron en los movimientos de aves o peces en busca de comida). El PSO puede verse como un algoritmo cooperativo simbiótico, porque los cambios de la posición que corresponden a cada partícula dentro del espacio de búsqueda se basan en la tendencia *psico-social* de las mismas que emulan el comportamiento de otra partícula.

El modelo de cúmulos de partículas está basado en una teoría sociocognitiva muy simple, que establece que el proceso de la adaptación cultural está formado por una componente de alto nivel y otro de bajo nivel. El primero de estos componentes se manifiesta mediante la formación de patrones emanados de la habilidad de los individuos para resolver problemas. La componente de bajo nivel se refiere al comportamiento de los individuos [27], que puede resumirse en tres principios: evaluar, comparar e imitar. Evaluar, es la tendencia a clasificar los estímulos como negativos o positivos, atractivos o repulsivos. Desde el punto de vista del aprendizaje puede definirse como

el cambio que permite al organismo mejorar la evaluación de su ambiente. Comparar e imitar se refiere a la comparación del individuo con respecto a sus vecinos para imitar sólo a aquellos que considere mejores a él mismo.

Algoritmo 6: Algoritmo básico propuesto IPSO.

```
1: iniciar_partículas(x)
2: evaluar_partículas(x)
3: repeat
4:    for i = 0 to nparticulas − 1 do
5:       localbest(x_i)
6:       for j = 0 to ndimensiones − 1 do
7:          x_i.vel_j =calcula_velocidad(x_{i,j})
8:       end for
9:    end for
10:   Gbest ← mejor_partícula(x)
11:   for i = 0 to nparticulas − 1 do
12:      for j = 0 to ndimensiones − 1 do
13:         if flip(P_perturbacion) then
14:            mutacion(x_{i,j})
15:         else if (x_{i,j} + vel_{i,j}) fuera de límite [ub,lb] then
16:            x_{i,j} = rnd[lb, ub]
17:         else
18:            x_{i,j} = x_{i,j} + vel_{i,j}
19:         end if
20:      end for
21:   end for
22:   for i = 0 to nparticulas − 1 do
23:      penaliza(x_i)
24:   end for
25: until Criterio de terminación
```

El procedimiento del algoritmo básico del PSO que proponemos en esta tesis se muestra en el algoritmo 6. Este algoritmo incorpora la mutación como operador de perturbación (recordemos que el PSO original no cuenta con un operador de mutación), buscando así que el algoritmo realice una mejor exploración del espacio de búsqueda.

En el algoritmo 6, x se refiere al cúmulo de partículas (la población) de tamaño *nparticulas*. Primero, se inicializan todas las partículas con valores aleatorios para sus *ndimensiones* variables de desición. Posteriormente, se entra a un ciclo en el cual se calcula, para cada partícula su velocidad. Una vez determinadas las aptitudes de todas las partículas se selecciona al mejor

de la población y se almacena en $G\vec{best}$. Luego, para cada partícula x_i se determina a su mejor vecino usando la función ($localbest$). En el siguiente paso, se determina, con una probabilidad $P_{perturbacion}$ si una partícula se mutará o no. Antes de agregar la velocidad a una partícula se verifica si el valor final no quedaría fuera de los rangos permisibles de las variables. En el algoritmo lb es la cota inferior y ub es la cota superior. Si al agregársele la velocidad a una partícula, ésta no se sale de los rangos permisibles, entonces la adición se permite; de lo contrario, la partícula adopta un valor aleatorio dentro del rango permisible de sus variables. En la parte final del algoritmo, procedemos a penalizar las soluciones que violen las restricciones usando la función $penaliza()$.

En el PSO, existen dos topologías básicas de interconexión de las partículas: la topología de anillo (modelo $lbest$) y la topología de interconexión completa (modelo $gbest$). A continuación, discutimos brevemente cada una de ellas.

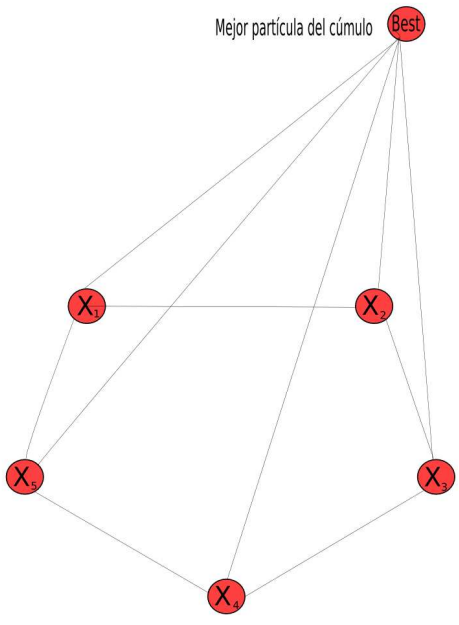

Figura 4.1: Topología del sistema propuesto.

En la topología de anillo (modelo *lbest*), cada partícula se comunica con n vecinos donde $n=2$. Esto quiere decir que cada partícula estará conectada a sus dos vecinos adyacentes, como se muestra en la figura 4.2. En contraste, en la topología de interconexión completa (modelo *gbest*) cada partícula estará conectada a la mejor partícula del cúmulo, como se muestra en la figura 4.3.

En la figura 4.1, se muestra la topología adoptada por nuestra aplicación. Podríamos decir que es una conexión híbrida ya que adopta una topología de anillo (modelo *lbest*) y una topología de interconexión completa (modelo *gbest*). De esta manera aseguramos una buena búsqueda y distribución de las partículas al calcular su nueva posición en el espacio de búsqueda (velocidad), logrando moverse en dirección de cualquiera de los atractores (*lbest* o *gbest*) o saltar a nuevas zonas del espacio de búsqueda, evitando así quedar atrapado en óptimos locales así como la convergencia prematura del algoritmo, debido al acelerado movimiento de las partículas hacia las zonas factibles.

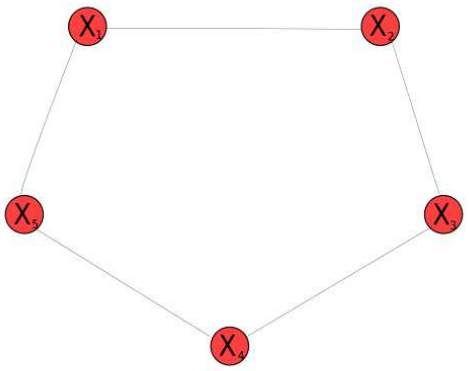

Figura 4.2: Topología *lbest*. Nótese que en este caso cada partícula tiene dos vecinos.

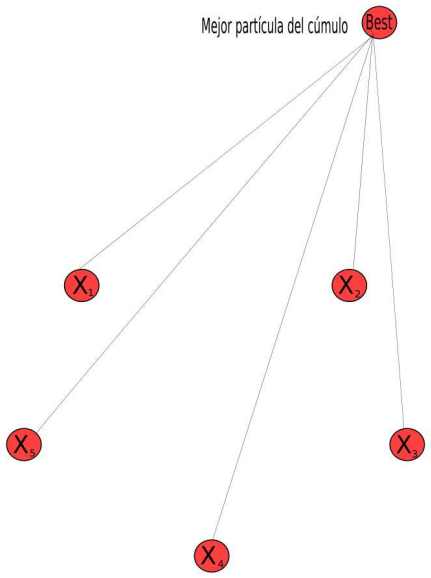

Figura 4.3: Topología *gbest*. Nótese que en este caso cada partícula está conectada al mejor del cúmulo.

En la función de velocidad del algoritmo del PSO se introdujo un esquema probabilístico, con la finalidad de tomar un camino, para determinar el atractor de una partícula $x_{i,j}$, como se muestra en el Algoritmo 7. Este funcionamiento se ilustra en la figura 4.4, donde la partícula A es atraída por sus vecinos B y G y la mejor partícula del cúmulo H, logrando así dar saltos en las direcciones de los atractores.

La idea principal de la técnica propuesta en esta tesis es poder salir de óptimos locales cuando la partícula *gbest* nos atrae hacia puntos de la zona factible que son falsos atractores y que nos impedirán llegar al óptimo global del problema. El Algoritmo 7 nos muestra el uso de atractores diferentes. En caso de que la partícula sea atraída por *gbest*, ésta es impulsada hacia la mejor partícula del cúmulo. En caso de que el atractor sea *lbest*, entonces es atraída hacia su vecindario. Por último, si la partícula es influenciada por

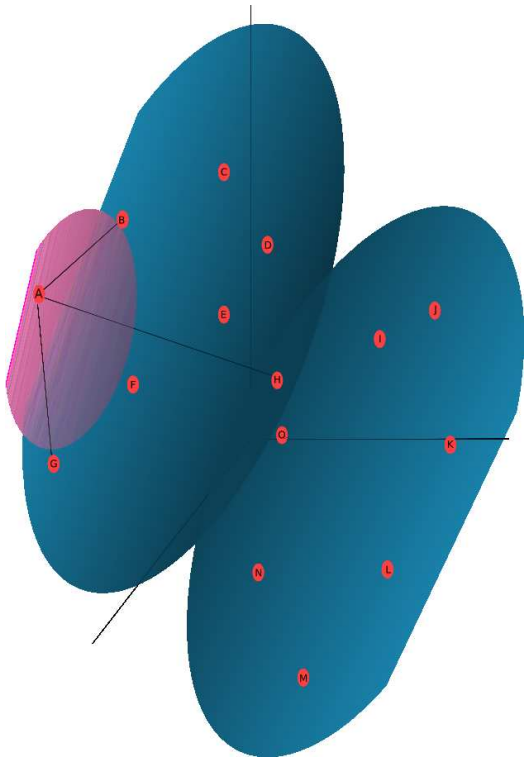

Figura 4.4: Ejemplo de atracción de las partículas que interfieren en el vuelo de una partícula.

sí misma, entonces dará un salto en función de lo que ha aprendido hasta el momento.

Para el caso de los valores R_1 y R_2, éstos deben encontrarse entre 0 y 1. La variación de estos dos parámetros influenciará el comportamiento del algoritmo en general. Por ejemplo si R_1 es muy grande, las partículas estarán influenciadas en gran medida por el líder. Si el parámetro es menor que 1 aseguraremos (dependiendo de qué tan pequeño sea R_1) que la partícula pueda estar sobrevolando el espacio entre el vecindario y la mejor partícula de la generación. El valor de R_2 también tiene un propósito importante, pues sirve para guiar la búsqueda por el vecindario y la búsqueda cognitiva de una partícula. Si la búsqueda se guía por el vecindario, con cierta probabilidad, esto buscará mejores soluciones entre estas dos partículas: el mejor local y la partícula que en ese momento se evalúa. Esto asegura un salto de *grano fino* en el espacio de búsqueda.

Los valores aleatorios [0,8, 1,0], [0,7, 1,0] y [0,5, 1,0], nos llevan a saltos en magnitudes pequeñas o grandes en términos del problema. La idea de modificar estos valores (originalmente propuestos por el algoritmo básico del PSO entre $[0, 1]$ para los tres casos), surge de la necesidad de impulsar más

la búsqueda hacia mejores lugares en la zona factible del problema. Si el vuelo es impulsado por *gbest*, al hacer crecer este valor buscamos mejores zonas con pasos de *grano grueso*, *grano intermedio* en el caso del *lbest* y de *grano fino* en caso de *pbest*. Cabe señalar que estos valores fueron calculados empíricamente y mostraron un mejor desempeño del algoritmo en general, en cuanto a: moderar la velocidad de las partículas, diversidad y evitar la convergencia prematura.

Algoritmo 7: Calcula la velocidad de una partícula x.

1: $W = rnd[0{,}1, 0{,}5]$
2: $C1 = rnd[1{,}5, 2{,}5]$
3: **if** flip(R_1) **then**
4: $\quad x_{i,j}.vel_{i,j} \leftarrow W \times vel_{i,j} + C1 * rand[0{,}8, 1{,}0] * (gbest_j - x_{i,j})$
5: **else if** flip(R_2) **then**
6: $\quad x_{i,j}.vel_{i,j} \leftarrow W \times vel_{i,j} + C1 * rand[0{,}7, 1{,}0] * (lbest_j - x_{i,j})$
7: **else**
8: $\quad x_{i,j}.vel_{i,j} \leftarrow W \times vel_{i,j} + C1 * rand[0{,}5, 1{,}0] * (pbest_j - x_{i,j})$
9: **end if**

El algoritmo 7 nos muestra la variable W la cual es calculada dentro de un rango [0.1,0.5], C_1 se obtiene de generar un número aleatorio dentro del rango [1.5,2.5] (líneas 1 y 2). Posteriormente tenemos el esquema de selección basado en probabilidades. Si R_1 es cierto entonces se calcula el vuelo con la fórmula de la línea 4 y el resultado de la operación es almacenada en la variable $x_{i,j}.vel_{i,j}$, de tal forma que, el vuelo es calculado con la partícula *gbest*. Si no, la velocidad es calculada con $lbest_{i,j}$. En caso contrario, la velocidad será calculada con $pbest_{i,j}$.

4.2. Manejo de restricciones

En esta sección presentamos el mecanismo adoptado para manejo de restricciones con el algoritmo del PSO. Este mecanismo está basado en [34] donde se diseñó un esquema de penalización para elegir a los líderes del cúmulo mediante el uso de reglas. Las reglas para seleccionar entre dos individuos son muy simples: si se comparan dos partículas factibles, se escoge a la que tenga la más alta aptitud. Si se compara una partícula factible con una infactible, se elige la partícula factible. Si se compara a dos partículas infactibles, se elegirá a la partícula que viole menos las restricciones del problema.

El proceso para penalizar consiste en normalizar las aptitudes de las partículas. Esto es, se extraen la peor y mejor aptitud para normalizar la aptitud de una partícula x_i de un cúmulo x. El proceso de penalización se

muestra en el Algoritmo 8 y se ilustra el funcionamiento del mismo en la figura 4.5.

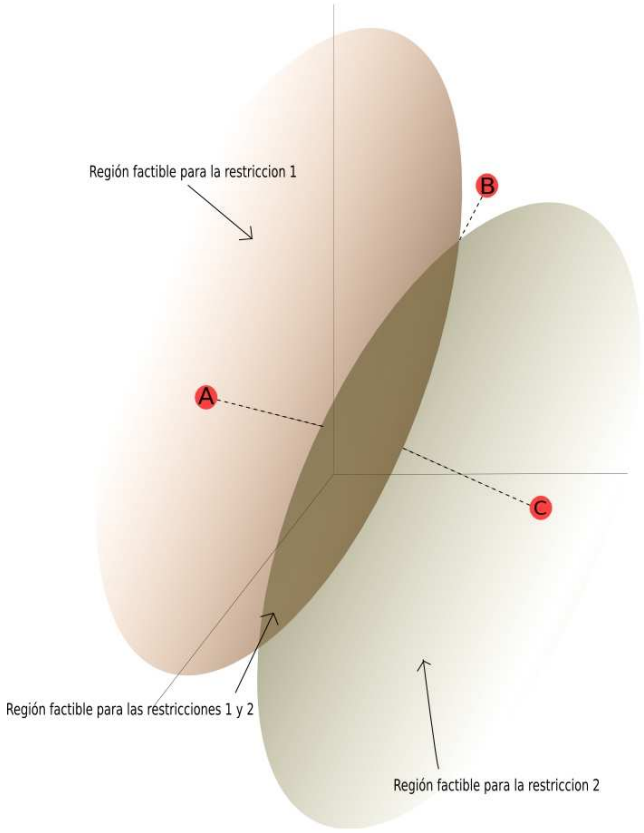

Figura 4.5: Esquema de manejo de restricciones.

Como podemos observar en el algoritmo 8, también las restricciones que una partícula viola son normalizadas. Esto es para tener un balance entre la normalización de la aptitud y el valor de la violación a fin de evitar que la penalización modifique de manera abrupta a la aptitud.
En el algoritmo 8, SUM es el valor de la normalización de las restricciones que son violadas, $min_{aptitud}$ es la mejor aptitud de la población, $max_{aptitud}$ es

Algoritmo 8: Penalización de la aptitud de una partícula x_i.

1: $SUM \leftarrow 0$
2: $min_{aptitud} \leftarrow minaptitud(x)$
3: $max_{aptitud} \leftarrow maxaptitud(x)$
4: $aptitud \leftarrow \frac{f(\vec{x_i}) - min_{aptitud}}{max_{aptitud} - min_{aptitud}}$
5: **for** $k = 0$ to $nrestricciones - 1$ **do**
6: **if** $x_i.restriccion_k > 0$ **then**
7: $SUM \leftarrow SUM + x_i.restriccion_k / Max.violacion_{[k]}$
8: **end if**
9: **end for**
10: $aptitud \leftarrow F * aptitud + SUM$

la peor aptitud de la población, F es un factor que nos indica el porcentaje del valor que se tomará de la aptitud de la partícula y $fitness$ es el valor de la partícula con su correspondiente penalización. $minfitness(x)$ es una función que extrae del cúmulo x la peor aptitud de las partículas y en su contraparte $maxfitness(x)$ extrae la mejor aptitud del cúmulo x. $Max.violacion_{[k]}$ es la máxima violación de la restriccion k.

En el algoritmo 8 observamos que $min_{aptitud}$ es la variable que almacena la mejor aptitud y $max_{aptitud}$ es la que almacena la peor aptitud del cúmulo x. Calculamos la aptitud de la partícula x_i mediante la fórmula expresada en la línea 4. Posteriormente, ciclamos de 0 hasta el número total de restricciones menos 1 para determinar qué partículas han violado alguna restricción y lo acumulamos en la variable SUM sumándole el valor de la violación entre la peor violación de esa misma restricción. Finalmente calculamos la aptitud de la partícula multiplicando el valor de aptitud por un factor F. Normalmente utilizamos el valor $F = 0{,}001$ (derivado empiricamente) y sumamos el valor del acumulador SUM.

4.3. Perturbación

En los algoritmos genéticos, la mutación es un operador secundario y se suele manejar con porcentajes bajos (típicamente entre *0.001* y *0.01*). No así para el PSO donde la mutación es el operador responsable de compensar la falta de capacidad explorativa que suele presentar la fórmula de velocidad en ciertos casos. El algoritmo mostrado en esta sección (algoritmo 9), es un operador de mutación similar al de los algoritmos genéticos al cual llamaremos *operador de perturbación* para el caso del PSO. Este operador demostró buen desempeño en las pruebas realizadas al algoritmo ya que el poder exploratorio de la mutación evita en gran medida que la búsqueda que-

de atrapada en óptimos locales y con una baja probabilidad de que pueda salir de ellos. De esta forma también aseguramos saltos de una partícula a espacios que no son alcanzados por el vuelo del propio PSO.

Los saltos en el espacio de búsqueda que realiza la perturbación podríamos considerarlos como saltos de grano grueso (junto con la variación drástica de los valores R_1 y R_2). El valor que se manejó para este parámentro en las pruebas realizadas fue de $P_{perturbacion} = 0{,}001$ (sin importar la cantidad de variables del problema). Algunos autores han propuesto diferentes valores para este parámetro. Por ejemplo, se ha sugerido usar $P_{perturbacion} = \frac{1}{L}$ (donde L es la longitud de la cadena cromosómica, que en el caso del PSO será el número de variables de decisión de cada problema en particular). Se sabe que este valor es el límite inferior para el porcentaje óptimo de este operador en el caso de algoritmos genéticos [1].

Algoritmo 9: Perturbación de una partícula $x_{i,j}$

```
1: if flip(P_perturbacion) then
2:     x_{i,j} = rnd[lb, ub]
3: end if
```

En el algoritmo 9, observamos un operador de mutación clásico de los algoritmos genéticos. Este metodo consiste en tener una probabilidad $P_{perturbacion}$ y haciendo un *flip* nos regresa un valor de 1 o 0, de esta manera decidimos si el vector $x_{i,j}$ va a almacenar un nuevo valor calculado dentro de los límites de la variable del problema $[lb, ub]$.

4.4. Algoritmo completo IPSO

A continuación mostramos el algoritmo completo de la técnica que proponemos en esta tesis. Para ello nos basaremos en los segmentos de código antes descritos.

A simple vista parecerían pocos los cambios con respecto al algoritmo mostrado en la sección 2.3, pero algunos de estos cambios son significativos. Por ejemplo, en el algoritmo 10, el vuelo de la partícula se calcula únicamente con el modelo *gbest* o *pbest*o *lbest* pero nunca se utiliza una combinación de éstos como en la versión original. Esto origina un vuelo más acotado con una intensidad en la velocidad menor a la del algoritmo original. Con esto se evita la convergencia prematura del algoritmo propuesto y se evita también que éste quede atrapado en óptimos locales. El vuelo puede ser en varias direcciones tanto hacia los atractores como también hacia posibles instancias intermedias del espacio de búsqueda. Se emplea también un esquema probabilístico para elegir hacia dónde seguirá el vuelo de una partícula del

Algoritmo 10: Algoritmo propuesto de nuestro sistema IPSO.

1: $\vec{Gbest} \leftarrow \vec{x_0}$
2: **for** $i = 0$ to $nparticulas - 1$ **do**
3: **for** $j = 0$ to $ndimensiones - 1$ **do**
4: $\vec{x_{i,j}} \leftarrow \vec{x_i} \leftarrow rnd(lb, ub)$
5: **end for**
6: $x_i.aptitud \leftarrow f(\vec{x_i})$
7: **if** $x_i.aptitud$ es mejor que $Gbest.aptitud$ **then**
8: $\vec{Gbest} \leftarrow \vec{x_i}$
9: $Gbest.aptitud \leftarrow x_i.aptitud$
10: **end if**
11: **end for**
12: **repeat**
13: **for** $i = 0$ to $nparticulas - 1$ **do**
14: $localbest_{i,j} \leftarrow Best_{fitness}(x_{i,j-1}, x_{i,j+1})$
15: **for** $j = 0$ to $ndimensiones - 1$ **do**
16: $W = rnd[0{,}1, 0{,}5]$
17: $C1 = rnd[1{,}5, 2{,}5]$
18: **if** flip(R_1) **then**
19: $vel_{i,j} \leftarrow W \times x_{i,j} + C1 * rand[0{,}8, 1{,}0] * (gbest_j - x_{i,j})$
20: **else if** flip(R_2) **then**
21: $vel_{i,j} \leftarrow W \times x_{i,j} + C1 * rand[0{,}7, 1{,}0] * (lbest_{i,j} - x_{i,j})$
22: **else**
23: $vel_{i,j} \leftarrow W \times x_{i,j} + C1 * rand[0{,}5, 1{,}0] * (pbest_{i,j} - x_{i,j})$
24: **end if**
25: **end for**
26: **end for**
27: $\vec{Gbest} \leftarrow$ mejor_partícula(x)
28: **for** $i = 0$ to $nparticulas - 1$ **do**
29: **for** $j = 0$ to $ndimensiones - 1$ **do**
30: **if** flip($P_{perturbacion}$) **then**
31: $x_{i,j} = rnd[lb, ub]$
32: **else if** $(x_{i,j} + vel_{i,j})$ fuera del límite [lb,ub] **then**
33: $x_{i,j} = rnd[ub, lb]$
34: **else**
35: $x_{i,j} = x_{i,j} + vel_{i,j}$
36: **end if**
37: **end for**
38: penaliza(x_i)
39: **end for**
40: **until** Criterio de terminación

cúmulo. Otro cambio importante es la incorporación de un mecanismo de perturbación que impulsa a la partícula a partes del espacio de búsqueda que no se pueden alcanzar mediante el vuelo, logrando de esta manera mantener diversidad en la población. En el siguiente capítulo validaremos la efectividad de estos cambios en el contexto de optimización con restricciones.

Capítulo 5

Resultados

En el presente capítulo mostraremos el funcionamiento de nuestro esquema propuesto del PSO para optimización global con restricciones. Para ello, adoptaremos las funciones de prueba reportadas en [36], para las cuales se describen sus características y los resultados obtenidos por nuestro esquema, así como su comparación con respecto con respecto a otras técnicas tres técnicas representativas del estado del arte en el área. A fin de presentar un análisis estadistico de resultados que sea más completo, se incluyen pruebas de normalidad de nuestros esquema y las gráficas de normalidad de algunas funciones de prueba.

5.1. Representación y operador genético utilizado

Para efectos de esta tesis se ha utilizado un solo tipo de representación, dada la naturaleza original del PSO que utiliza representación real (aunque con adaptaciones puede llegar a operar con una representación binaria [27]).

Para la representación real se utilizó un operador de mutación probabilística, el cual consiste en cambiar aleatoriamente el valor de una variable de decisión de una partícula, al igual que en los algoritmos genéticos.

5.2. Resultados

La implementación del algoritmo prupuesto fue realizada en *lenguaje C*. El compilador utilizado fue *g++* versión 3.3.5 bajo el sistema operativo Linux. La distribución de Linux utilizada fue Debian. Los parámetros para todas las funciones son los siguientes:

- 40 partículas.

- 8750 generaciones.
- $R_1 = 0{,}9$.
- $R_2 = 0{,}3$.
- $P_{perturbacion} = 0{,}001$.

Cabe mencionar que los valores que se utilizaron para R_1 y R_2 y $P_{perturbacion}$ fueron derivados empíricamente. C_1, C_2 y W fueron adoptados según se sugiere en [34].

Los resultados de las trece funciones de prueba se presentan de la siguiente manera:

1. Tabla con las estadísticas de las trece funciones.
2. Tabla con estadísticas con diferentes técnicas evolutivas, las cuales utilizaron las mismas funciones de prueba, de tal forma que podamos comparar el desempeño de nuestro esquema.
3. Discusión de los resultados obtenidos.
4. Estadísticas de normalidad e intervalos de confianza de cada función.
5. Gráficas de normalidad de las funciones para aquellos casos en que la hipótesis sea cierta.

5.2.1. Estadísticas de las funciones de prueba

Se realizaron 30 corridas de cada función a efecto de poder evaluar su desempeño estadísticamente. En la tabla 5.1 se muestran los resultados obtenidos por nuestro algoritmo para las trece funciones de prueba. Como en todos los casos se adoptó una población de 40 partículas y un total de 8750 generaciones por corrida, nuestro algoritmo realizó 350,000 evaluaciones por corrida por problema. Este es el mismo número de evaluaciones realizadas por la jerarquización estocástica [36], que es el método más competitivo de la actualidad, y contra el cual comparamos resultados.

5.2.2. Discusión

Para validar nuestros resultados, los comparamos con respecto a dos algoritmos representativos del estado del arte en el área: la jerarquización estocástica [36] y los mapas homomorfos [29]. Con la finalidad de comparar resultados con respecto a otro esquema de manejo de restricciones basado

F	Óptimo	Mejor	Media	Mediana	Peor	Desv Est
g01	**-15.0**	**-15.00000**	-15.00000	-15.00000	-15.00000	0.00000
g02	**-0.803619**	-0.803580	-0.71263	-0.72265	-0.58446	0.05813
g03	**-1.0**	**-1.00044**	-0.99818	-1.00003	-0.95814	0.00783
g04	**-30665.539**	-30665.537191	-30665.51754	-30665.52942	-30665.25872	0.05030
g05	**5126.498**	5126.50133	5431.40594	5213.82906	6112.21636	379.74832
g06	**-6961.814**	**-6961.85420**	-6961.09626	-6961.15097	-6959.44958	0.58359
g07	**24.306**	24.42530	26.18767	25.86329	29.82210	1.60361
g08	**-0.095825**	**-0.095825**	-0.095825	-0.095825	-0.095825	0.00000
g09	**680.63**	680.65552	680.72376	680.69860	680.78529	0.05678
g10	**7049.25**	7064.05042	8075.66568	7870.81864	11501.80975	1019.12959
g11	**0.75**	**0.749999**	0.75001	0.75000	0.75003	0.00001
g12	**-1.0**	**-1.00000**	-1.00000	-1.00000	-1.00000	0.00000
g13	**0.053950**	0.05489	0.48544	0.44699	1.00000	0.30018

Tabla 5.1: Estadísticas de las treinta corridas para cada una de las funciones de nuestro esquema propuesto IPSO.

		Mejor		Media		Peor	
F	Óptimo	IPSO	SR	IPSO	SR	IPSO	SR
g01	**-15.0**	**-15.00000**	-15.00000	**-15.00000**	-15.0000	**-15.00000**	-15.00000
g02	**-0.803619**	**-0.803580**	-0.803515	-0.71263	-0.781975	-0.58446	-0.726288
g03	**-1.0**	**-1.00044**	-1.0	-0.99818	-1.0	-0.95814	-1.0
g04	**-30665.539**	**-30665.537191**	-30665.539	-30665.51754	-30665.539	-30665.25872	-30665.539
g05	**5126.498**	5126.50133	5126.497	5431.40594	5128.881	6112.21636	5142.472
g06	**-6961.814**	**-6961.854202**	-6961.814	**-6961.09626**	-6875.940	**-6959.44958**	-6350.262
g07	**24.306**	24.42530	24.307	26.18767	24.374	29.82210	24.642
g08	**-0.095825**	**-0.095825**	-0.095825	**-0.095825**	-0.095825	**-0.095825**	-0.095825
g09	**680.630**	680.660201	680.630	680.72376	680.656	680.78529	680.763
g10	**7049.25**	7064.05042	7054.316	8075.66568	7559.192	11501.80975	8835.665
g11	**0.75**	**0.749999**	0.75000	**0.75001**	0.75001	**0.75003**	0.7500
g12	**-1.0**	**-1.00000**	-1.00000	**-1.00000**	-1.00000	**-1.00000**	-1.00000
g13	**0.053950**	0.05489	0.053945	0.48544	0.067543	1.00000	0.216915

Tabla 5.2: Comparación de nuestro algoritmo IPSO con respecto a la jerarquización estocásticas (SR) [36].

		Mejor		Media		Peor	
F	Óptimo	IPSO	HM	IPSO	HM	IPSO	HM
g01	**-15.0**	**-15.00000**	-14.788600	**-15.00000**	-14.708200	**-15.00000**	-14.615400
g02	**-0.803619**	**-0.803580**	-0.790406	-0.71263	-0.796710	-0.58446	-0.791190
g03	**-1.0**	**-1.00044**	-0.999700	-0.99818	-0.998900	-0.95814	-0.997800
g04	**-30665.539**	**-30665.537191**	-30664.5000	**-30665.51754**	-30655.300	**-30665.25872**	-30645.900000
g05	**5126.498**	**5126.50133**	NA	**5431.40594**	NA	**6112.21636**	NA
g06	**-6961.814**	**-6961.854202**	-6952.10	**-6961.09626**	-6342.60000	**-6959.44958**	-5473.900000
g07	**24.306**	**24.42530**	24.62	26.18767	24.826000	29.82210	25.069000
g08	**-0.095825**	**-0.095825**	-0.95825	**-0.095825**	-0.089157	**-0.095825**	-0.029144
g09	**680.630**	**680.660201**	680.910000	**680.72376**	681.160000	**680.78529**	683.180000
g10	**7049.25**	**7064.05042**	7147.900000	**8075.66568**	8163.600000	11501.80975	9659.300000
g11	**0.75**	**0.749999**	0.750000	**0.75001**	0.750000	**0.75003**	0.750000
g12	**-1.0**	**-1.00000**	-1.000000	**-1.00000**	-0.999135	**-1.00000**	-0.991950
g13	**0.053950**	**0.05489**	NA	**0.48544**	NA	**1.00000**	NA

Tabla 5.3: Comparación de nuestro algoritmo IPSO con respecto a los mapas homomorfos (HM) [29].

		Mejor		Media		Peor	
F	**Óptimo**	IPSO	TC	IPSO	TC	IPSO	TC
g01	**-15.0**	**-15.000**	-15.000	**-15.000**	-15.000	**-15.000**	-15.000
g02	**-0.803619**	**-0.803580**	-0.803432	-0.71263	-0.790406	-0.58446	-0.750393
g03	**-1.0**	**-1.00044**	-1.004720	-0.99818	-1.003814	-0.95814	-1.002490
g04	**-30665.539**	**-30665.537191**	-30665.5000	**-30665.51754**	-30665.5000	-30665.25872	-30665.5000
g05	**5126.498**	**5126.50133**	5126.6400	5431.40594	5461.081333	6112.21636	6104.7500
g06	**-6961.814**	**-6961.854202**	-6961.8100	-6961.09626	-6961.8100	-6959.44958	-6961.8100
g07	**24.306**	24.42530	24.351100	26.18767	25.355771	29.82210	27.316800
g08	**-0.095825**	**-0.095825**	-0.95825	**-0.095825**	-0.095825	**-0.095825**	-0.95825
g09	**680.630**	680.660201	680.6380	**680.72376**	680.852393	680.78529	681.5530
g10	**7049.25**	7064.05042	7057.59	8075.66568	7560.047857	11501.80975	8104.3100
g11	**0.75**	**0.749999**	0.749999	**0.75001**	0.750107	**0.75003**	0.752885
g12	**-1.0**	**-1.000**	-1.000	**-1.000**	-1.000	**-1.000**	-1.000
g13	**0.053950**	**0.05489**	0.068665	**0.48544**	1.716426	**1.00000**	13.669500

Tabla 5.4: Comparación de nuestro algoritmo IPSO con respecto a los resultados reportados por Toscano-Coello (TC) [34].

en el PSO, también incluímos una comparación con respecto al método de Toscano y Coello [34].

Se hace notar que el método de Toscano y Coello [34] realiza 350,000 evaluaciones de la función de aptitud, al igual que la jerarquización estocástica y nuestro método. Los mapas homomorfos, sin embargo, se realizaron 1,400,000 evaluaciones de la función de aptitud.

En la tabla 5.1 resume las estadísticas realizadas a las trece funciones de experimentación. La tabla muestra el mejor valor "óptimo conocido" para cada problema junto con las estádisticas ya mencionadas anteriormente.

En la tabla 5.1 muestra los resultados de nuestra técnica, a la cual llamaremos "Improved Particle Swarm Optimization" (IPSO). Para cada función se muestra su óptimo o mejor conocido.

Se hace notar que las restricciones de igualdad de la forma:

$$h(\vec{x}) = 0 \tag{5.1}$$

fueron transformadas en desigualdades de la forma:

$$-\epsilon \leq |g(\vec{x})| \leq \epsilon \tag{5.2}$$

Este tipo de transformación ha sido adoptado por otros autores (p. ej., [36]). En nuestro caso, $\epsilon = 1 \times 10^{-5}$ en todos nuestros experimentos.

De la tabla 5.1, podemos decir varias cosas acerca del comportamiento de nuestro algoritmo. Primero, queremos resaltar que en *g03* y *g11* presentamos un resultado ligeramente mejores al óptimo debido al valor de ϵ adoptado para las restricciones de igualdad. De aplicarse rigurosamente las restricciones de igualdad en esos problemas, dichas soluciones no serían posibles.

En *g06* ocurre algo similar, pues nuestra mejor solución viola muy ligeramente una de sus restricciones activas (con un valor de 1×10^{-5}). De ahí que nuestra solución parezca mejorar al óptimo.

De la tabla 5.1 puede verse también que nuestro algoritmo alcanza el óptimo en su mejor resultado en siete problemas: *g01, g04, g06, g08, g11* y *g12*. Además, converge muy cerca del óptimo en los problemas restantes. En el apéndice A mostramos las gráficas de convergencia para cada función y en el apéndice B mostramos los valores de las variables de decisión correspondientes al mejor resultado obtenido por nuestro algoritmo para cada problema, así como los valores de las restricciones en cada caso. Excepto por *g05, g10* y *g13*, nuestro algoritmo muestra desviación estándar bastante bajas, lo que indica un comportamiento bastante consistente y robusto.

El comportamiento de nuestro algoritmo en los casos de excepción lo atribuimos al factor movimiento aleatorio con que cuenta nuestro algoritmo para los casos en que nos saliamos del rango de una variable, así como a la dificultad para satisfacer las restricciones de esas funciones (*g05, g10 y g13*).

Uno de los problemas que nos causó muchas dificultades fue *g02*, pues a pesar de que cuenta con una zona factible muy grande, tiene muchos óptimos locales en los que un algoritmo puede quedar fácilmente atrapado. En las primeras versiones de nuestro algoritmo, este problema hacia ver el peor comportamiento de nuestro algoritmo debido a que éste no contaba con una buena capacidad explorativa. Ese fue lo que motivó el uso de un operador de perturbación, el cual, resultó ser benéfico también en el resto de los problemas.

Algo interesante sobre *g02* es que el uso de valores más grandes para el porcentaje de mutación (p.ej. $P_{perturbacion}$ =0.01) mejora nuestros resultados. Sin embargo, al final decidimos adoptar $P_{perturbacion}$ =0.001 como un compromiso que proporciona resultados aceptables para todas las funciones de prueba.

Los dos problemas en los que nuestro algoritmo tuvo más dificultades fueron *g05* y *g13*. En *g05*, nuestro método convergío a la zona factible únicamente en 5 de las treinta corridas realizadas. En *g13*, sólo convergió en 2 de las 30 corridas realizadas. En la tabla 5.2 comparamos nuestro algoritmo (IPSO) con respecto a la jerarquización estocástica [36]. Se han marcado en **negritas** aquellos resultados en los que nuestro algoritmo igualó o mejoró a la jerarquización estocástica tanto en mejor resultado, como en la media y el peor. A partir de estos resultados puede verse que nuestro algoritmo resultó bastante competitivo con respecto al mejor método de manejo de restricciones que se tiene hasta ahora.

Al comparar nuestro IPSO con respecto a los mapas homomorfos [29] en la tabla 5.3, vemos que nuestro algoritmo es el claro ganador. Nuevamente, usamos **negritas** para mostrar los casos en los que nuestro algoritmo igualó o mejoró los resultados de los mapas homomorfos. Se hace notar que los mapas homomorfos no produjeron resultados factibles en *g05* y que no incorporaron

a *g13* dentro de su conjunto de funciones de prueba. Los mapas homomorfos no pudieron mejorar nuestro "mejor" resultado en ningún caso, y sólo produjeron mejores resultados en la media y el peor en *g02, g03* y *g07*.

La tabla 5.4 muestra la comparación de resultados entre nuestro algoritmo y el método (basado en PSO) de Toscano y Coello (TC) [34]. Una vez más, usamos **negritas** para mostrar los casos en los que igualamos o mejoreamos los resultados de TC. Puede verse que nuestro algoritmo encuentra un mejor resultado que TC en 10 de las funciones de prueba (*g01, g02, g03, g04, g05, g06, g06, g08, g11, g12* y *g13*). En las soluciones promedio, TC supera a nuestro método en seis funciones de prueba (*g02, g03, g05, g06, g07* y *g10*). Es en la peor solución encontrada en donde TC se comporta mejor que nuestro método, pues lo supera en ocho funciones (*g02, g03, g04, g05, g06, g07, g08, g09* y *g10*).

Claramente pudimos observar, el esquema propuesto puede lidiar con problemas con restricciones moderadas (*g04*), problemas alto número de restricciones. Con baja dimesionalidad (*g06,g08*), moderada (*g09*) y alta (*g01,g02,g03,g07*). Con diferentes tipos de restricciones y sus combinaciones (lineales, no lineales, igualdad y desigualdad), regiones factibles grandes (*g02*), muy pequeñas (*g05 y g13*) o disjuntas *g12*. También el algoritmo es capaz de lidiar con espacios de búsqueda grandes y regiones factibles pequeñas aunque con menor robustez (*g10*), dado que, el poder exploratorio se centra en una búsqueda un tanto no dirijida hacia la mejor partícula del cúmulo. Sin embargo nuestro algoritmo es capaz de lidiar con problemas donde el óptimo global se encuentra en la frontera de la región factible (*g01, g02, g04, g06, g07 y g09*).

5.3. Análisis de resultados

A fin de poder determinar la confiabilidad estadística de nuestros resultados, el primer paso es verificar la distribución de nuestras soluciones. Si ésta se ajusta a la normal se usa la siguiente expresión para calcular los intervalos de confianza de cada función:

$$x \pm t \times \left(\frac{s}{\sqrt{n}} \right) \tag{5.3}$$

donde x es la media de la muestra, t es el valor crítico de la distribución que puede ser encontrado en tablas, n es el tamaño de la muestra ($n = 30$, en nuestro caso). Y s es la desviación estándar de la muestra. Por ejemplo, para una confiabilidad del 95 %, t =1.697. Sin embargo, como en nuestro caso la

distribución de nuestras soluciones no se ajusta a la normal, debemos recurrir al análisis no paramétrico.

Entre las pruebas no paramétricas que comúnmente se utilizan para verificar si una distribución se ajusta o no a una distribución esperada (en particular a la distribución normal), se encuentra la prueba de Kolmogorov-Smirnov (K-S). Esta prueba es muy potente con muestras grandes $n \geq 30$. El nivel de medición de la variable y su distribución son elementos que intervienen en la selección de la prueba que se utilizará en el procedimiento posterior. De hecho, si la variable es continua con distribución normal, se podrán aplicar técnicas paramétricas. Si es una variable discreta o continua no normal, sólo son aplicables técnicas no paramétricas pues aplicar las primeras arrojaría resultados de dudosa validez.

La prueba K-S de una muestra es una prueba de bondad de ajuste. Esto es, intentar hallar el grado de similitud entre la distribución de un conjunto de valores de la muestra y alguna distribución teórica específica. La prueba K-S determina si se puede pensar razonablemente que las mediciones muestrales provienen de una población que tenga esa distribución teórica. En la prueba se compara la distribución de frecuencia acumulativa de la distribución teórica con la distribución de frecuencia acumulativa observada y se determina el punto en el que estas dos distribuciones muestran la mayor divergencia.

Para realizar esta prueba nos apoyamos en el software de prueba llamado DataLab versión 2.12, Lohninger, 1992-2001, el cual nos arroja el valor K-S y una tabla de valores críticos dependiente del número de muestras de la población. En nuestro caso, $n = 30$.

5.3.1. Resultados de la prueba K-S

Se realizaron las pruebas de Kolmogorov-Smirnov y se obtuvo la tabla 5.5. Se hace notar que para el caso en que una muestra siempre arroje el mismo valor (p. e. si llega siempre al óptimo global), no es necesario realizar la prueba de Kolmogorov-Smirnov porque la muestra no tiene una distribución normal. Por ejemplo, la figura 5.1 muestra la función *g01* donde apreciamos que las soluciones siempre fueron $f(\vec{x}) = -15{,}0$ que es el óptimo global de la función de prueba. El mismo caso presentan las funciones *g08* y *g12*.

Cuando la prueba Kolmogorov-Smirnov (ver tabla 5.5), arroja un valor mayor al mostrado en la tabla 5.6 (para la confiabilidad deseada) se puede decir que la muestra no se ajusta a una distribución normal. En caso contrario, si el valor de la prueba Kolmogorov-Smirnov no rebasa los valores mostrados en la tabla de valores critícos, entonces decimos que la muestra se ajusta a la normal con el porcentaje de certeza adoptado.

Problema	Prueba K-S
g01	0.4326
g02	0.0997
g03	0.3745
g04	0.3745
g05	0.4425
g06	0.1583
g07	0.1395
g08	0.5382
g09	0.2004
g10	0.1595
g11	0.2680
g12	0.8413
g13	0.2742

Tabla 5.5: Resultados de la prueba Kolmogorov-Smirnov de las trece funciones.

Confiabilidad (%)	Valor crítico
99	0.2928
95	0.2443
90	0.2191
85	0.2048
80	0.1922

Tabla 5.6: Porcentaje de confiabilidad y valores críticos para una muestra de treinta soluciones.

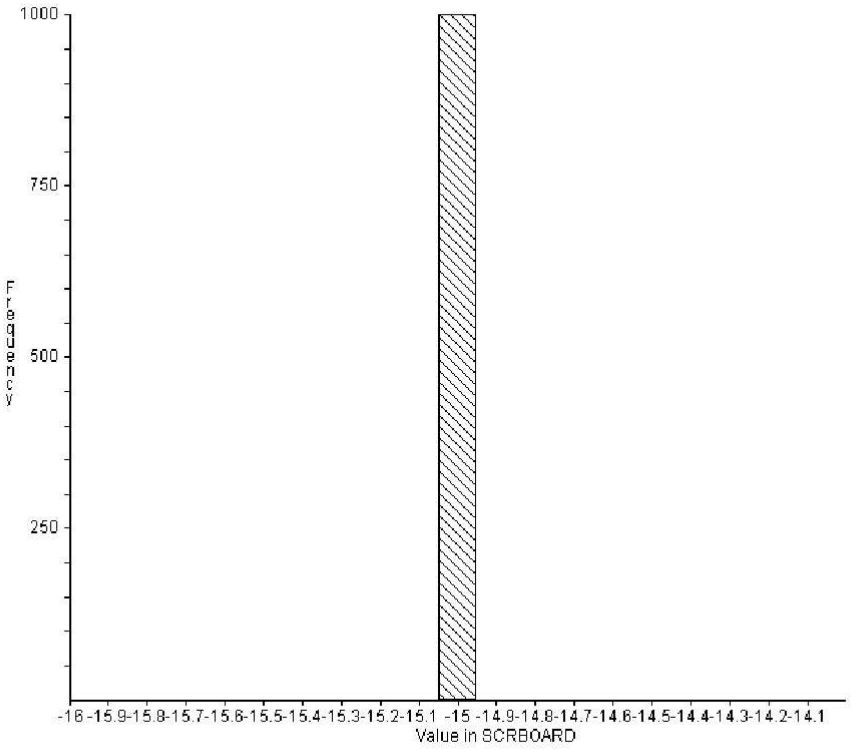

Figura 5.1: Histograma para *g01*

En nuestro caso, adoptamos una confiabilidad del 95 %, por lo que nuestro valor crítico es 0.2443 para la prueba K-S. Viendo la tabla 5.5, encontramos cinco funciones para las cuales la prueba K-S arroja valores por abajo del crítico son cinco: g02, g06, g07, g09 y g10. Los histogramas de estas funciones se muestran en las figuras 5.2, 5.3, 5.4, 5.5 y 5.6. Como en estos casos se ajustan a una distribución normal, el intervalo de confianza se calculó con la

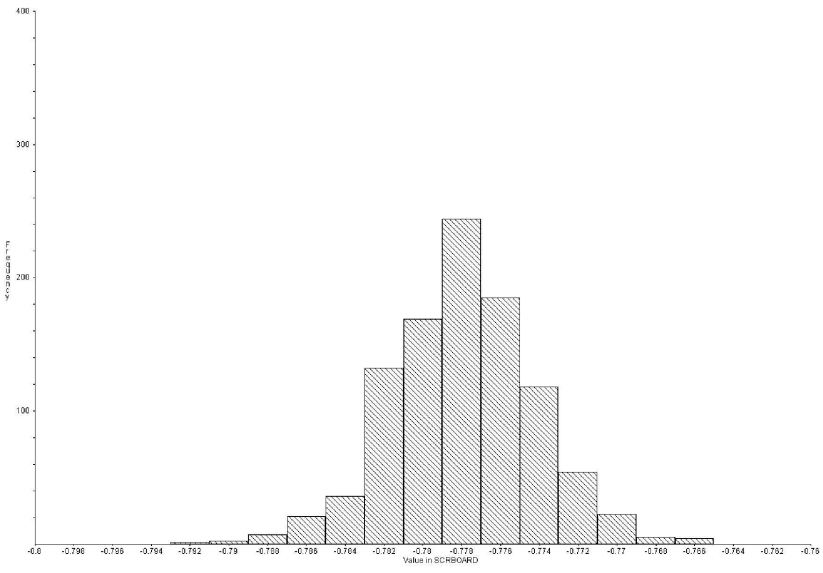

Figura 5.2: Histograma para *g02*.

ecuación 5.3. La tabla 5.7 muestra los intervalos de confianza de estas cinco funciones. Para determinar estos intervalos de confianza, se usó $t = 1{,}697$ (para una confiabilidad del 95 %) y $n = 30$ (el tamaño de la muestra).

5.3.2. Bootstrap

El bootstrap es un método estadístico no paramétrico utilizado para establecer si una muestra de tamaño n es representativa, es decir, qué tanto cambia una muestra obtenida con respecto a otra distribución deseada. Este método calcula un intervalo de confianza mediante el cual podemos asegurar

F	**Intervalo de confianza**
g02	[-0.69462,-0.73064]
g06	[-6960.97016, -6961.33178]
g07	[26.68451, 25.69083]
g09	[680.74135,680.70617]
g10	[8391.42096, 7759.910397]

Tabla 5.7: Intervalos de confianza encontrados con la formula 5.3 .

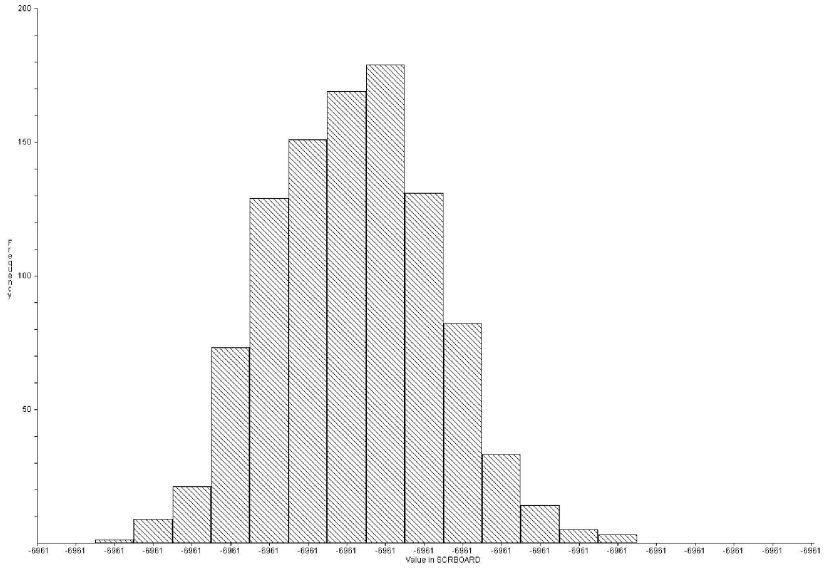

Figura 5.3: Histograma para *g06*.

que si tomamos una muestra de nuestro universo (en este caso un conjunto de soluciones dadas por el algoritmo propuesto), dicha muestra estará dentro del intervalo de confianza calculado. Entre más cercano este el intervalo de confianza al óptimo y entre más cerrado sea dicho intervalo, más eficiente será nuestra técnica empleada.

Las muestras se recopilaron de la siguiente manera: de las funciones de prueba se realizaron treinta ejecuciones y de esta manera obtuvimos treinta soluciones para cada una de las funciones de prueba. Pero si queremos saber la representatividad de nuestra muestra obtenida, es decir, si obtuviéramos otras treinta soluciones más ¿qué tan diferentes serían estas muestras? Para saberlo, es necesario realizar la prueba del bootstrap [8]. Esto se hace extrayendo un gran número de muestras de tamaño n (para el caso de estudio se obtuvieron 30) de la muestra original aleatoriamente y con reposición. Asi, aunque cada remuestra (muestra de la muestra) tendrá el mismo número de elementos que la muestra original, mediante el remuestreo con reposición cada remuestra podría tener algunos de los datos originales representados en ella más de una vez, y algunos que no aparecieran. Por lo tanto, cada una de estas remuestras probablemente será levemente y aleatoriamente diferente de la muestra original.

La prueba de bootstrap se realizó con el programa *Resampling Stats* ver-

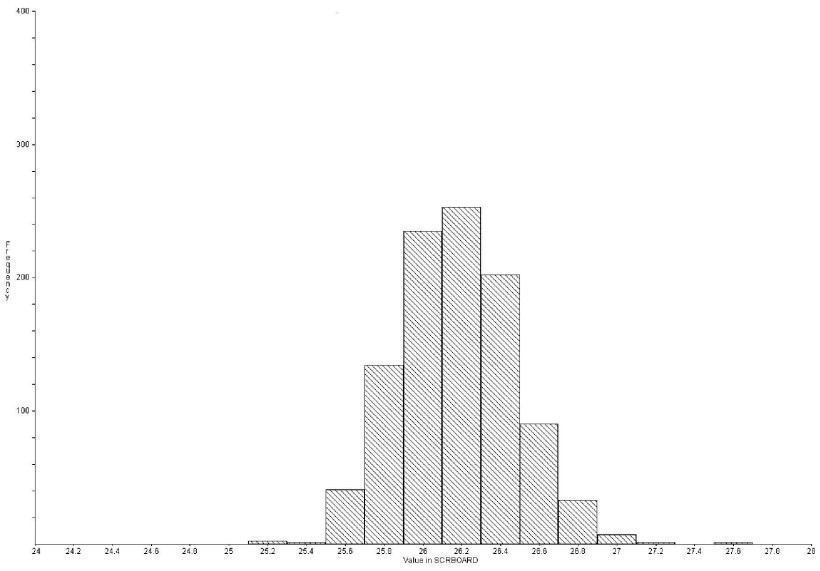

Figura 5.4: Histograma para *g07*.

sión 5.0.2, Resampling Stats Inc. 1990-1999. Fue necesario escribir un programa (ver algoritmo 11) que nos diera el manejo y la salida de los datos de acuerdo a nuestras necesidades.
READ lee de un archivo ASCII los datos y los almacena en un vector A. Posteriormente, ciclamos mil veces. SAMPLE toma treinta soluciones contenidas en A de manera aleatoria y las coloca en un nuevo vector D de donde se obtiene la media la cual se resguarda en la variable C y más tarde, en el vector BOARD se van almacenando las distintas medidas obtenidas en cada una de las mil iteraciones. Con PERCENTILE aplicado al vector BOARD se obtienen los intervalos de confianza del bootstrap que nos permitirán saber si nuestra muestra es representativa. Entre paréntesis colocamos el intervalo que indica a PERCENTILE que coloque los intervalos sin tomar en cuenta las peores ni las mejores 2.5 % medias contenidas en BOARD. Con esto obtenemos una confiabilidad del 95 %, que es la que se utiliza comúnmente en la practica [8]. PRINT imprime en pantalla el intervalo y con WRITE lo escribimos en un archivo, utilizando APPEND para que no se borre el contenido anterior del archivo si es que existiera información.

Los intervalos de confianza obtenidos por el bootstrap se muestran en la tabla 5.8

Algoritmo 11: Programa de entrada para el software Resampling Stats

```
READ FILE "MUESTRA1.DAT" A
REPEAT 1000
SAMPLE 30 A B
MEAN B C SCORE C BOARD
END
HITOGRAM BOARD
PERCENTILE BOARD (2.5 97.5) INTERVAL
PRINT INTERVAL
WRITE FILE "BOOTSTRAP.DAT" APPEND INTERVAL
```

F	**Intervalo de confianza**
g01	[-15.0, -15.0]
g03	[-0.995126, -1.00004]
g04	[-30665.5, -30665.5]
g05	[5128.08, 5271.76]
g08	[-0.095825, -0.095825]
g11	[0.750003, 0.750008]
g12	[-1, -1]
g13	[0.329299, 0.71056]

Tabla 5.8: Intervalos de confianza encontrados con el bootstrap.

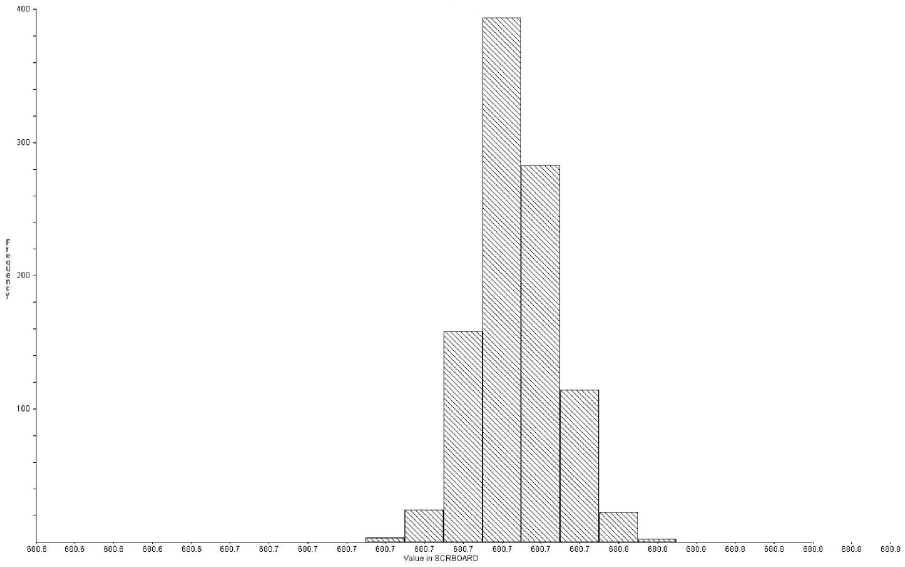

Figura 5.5: Histograma para *g09*.

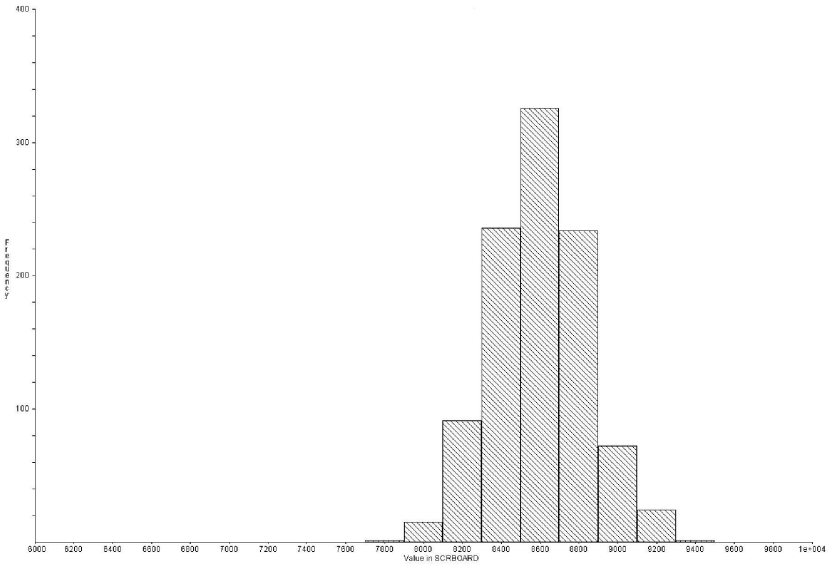

Figura 5.6: Histograma para *g10*.

Capítulo 6

Conclusiones y trabajo futuro

En esta tesis se ha propuesto un nuevo mecanismo de manejo de restricciones para la heurística denominada PSO, a la cual llamamos IPSO. El esquema propuesto, aunque aparentemente simple, es producto de un estudio minucioso del comportamiento del PSO en optimización global con restricciones. Entre los mecanismos propuestos, los más importantes son el operador de mutación y el esquema de vecindarios adoptado.

El método propuesto fue comparado con respecto a tres técnicas representativas del estado del arte en optimización con restricciones [36, 29], incluyendo una basada en PSO [34].

Los resultados obtenidos indican que el PSO es un motor de búsqueda con muy buen potencial para optimización global con restricciones, pues su desempeño resultó muy competitivo.

Como parte de nuestro trabajo futuro, planeamos usar un mecanismo más elaborado para la selección de lideres, el cual favorezca a individuos infactibles que se encuentren muy cerca de la zona factible. Creemos que eso mejorará el desempeño del algoritmo, sobre todo en los problemas en los cuales tiene muchas dificultades para llegar a la zona factible.

En esa misma dirección, nos interesa explorar mecanismos alternativos para mantener diversidad, así como nuevas topologias de vecindarios.

Finalmente, creemos importante realizar un análisis de varianza a fin de determinar la sensibilidad del algoritmo a sus parámetros. Esto podría ayudarnos a diseñar un mecanismo de ajuste automático de parámetros que facilitaría de sobremanera el uso del algoritmo.

Apéndice A

En el presente apéndice mostramos las gráficas de convergencia para cada uno de los problemas expuestos en la sección 3.5.1. Cada figura corresponde a la muestra obtenida más cercana a la mediana de nuestros resultados.

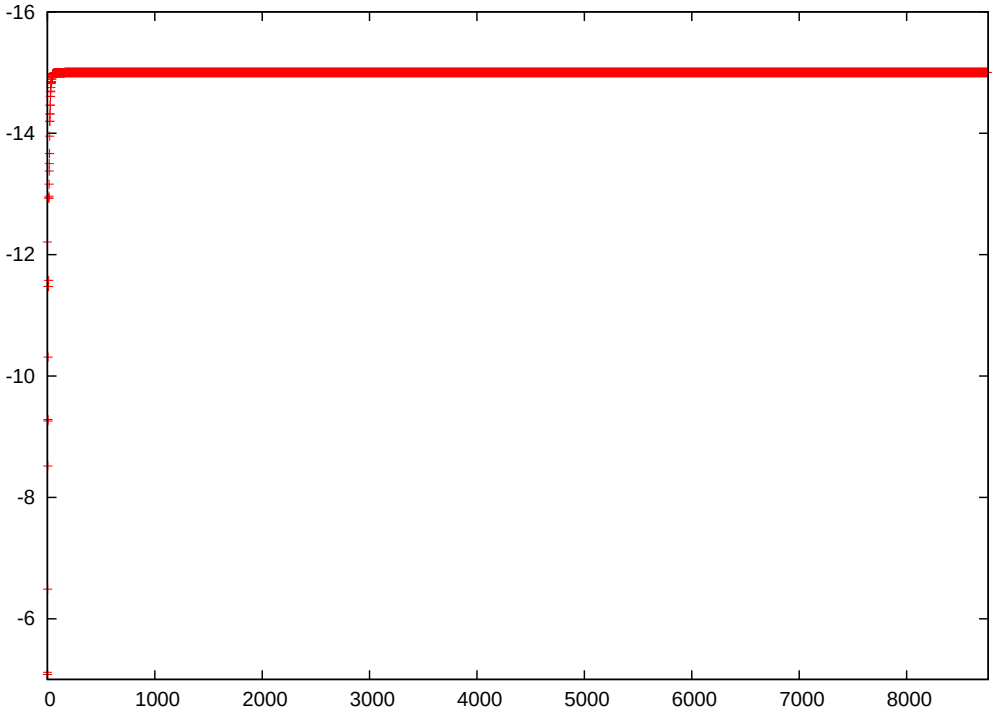

Figura A.1: Gráfica de convergencia para la función *g01*.

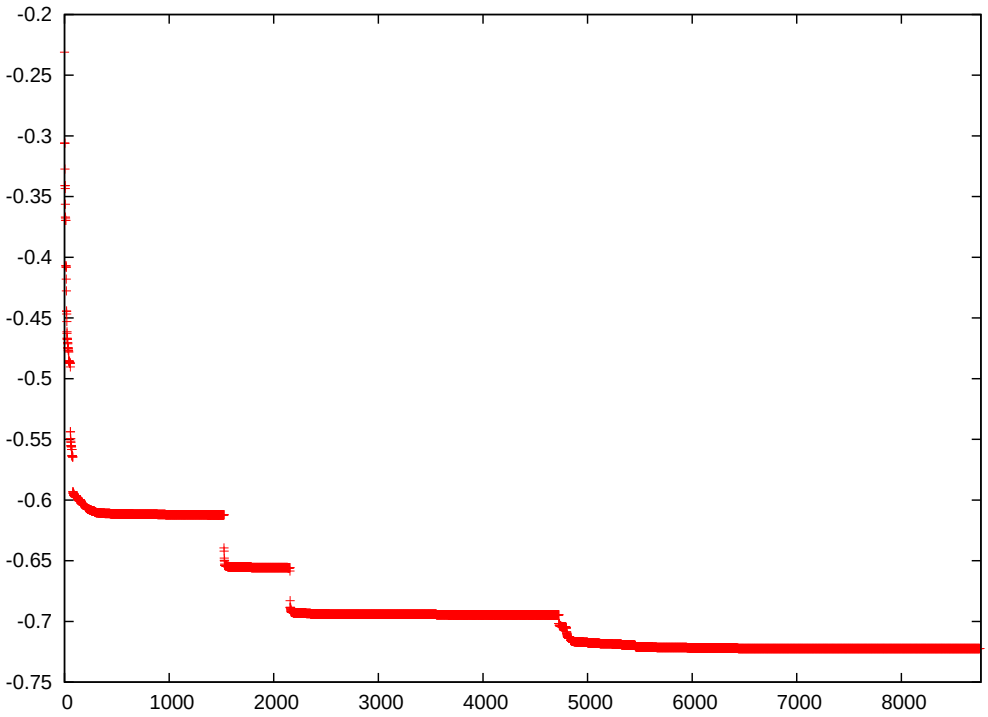

Figura A.2: Gráfica de convergencia para la función *g02*.

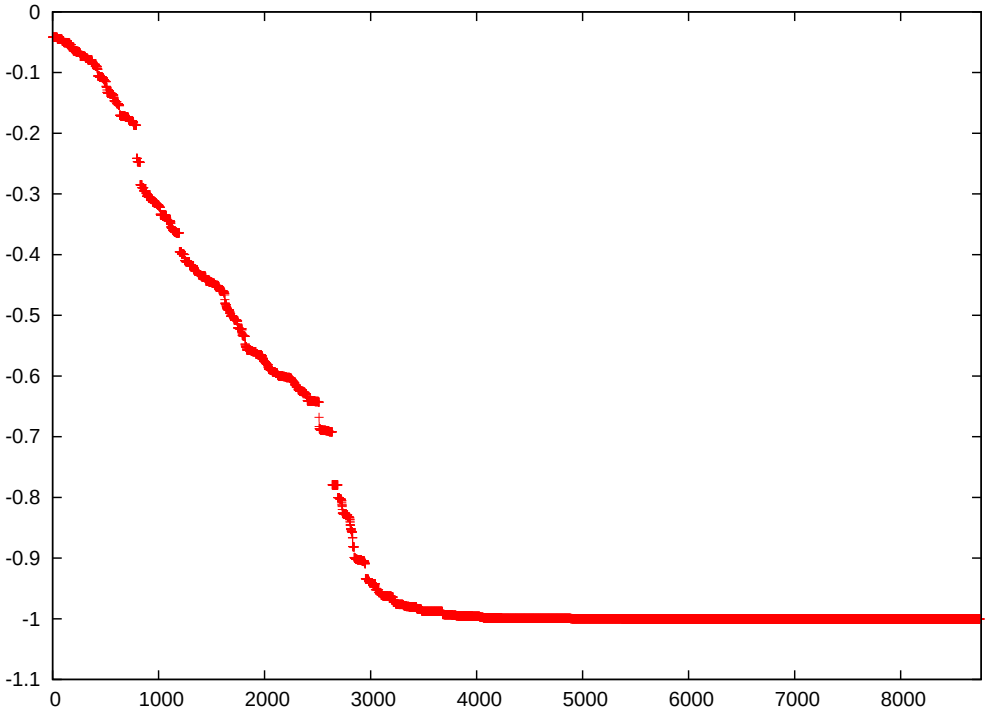

Figura A.3: Gráfica de convergencia para la función *g03*.

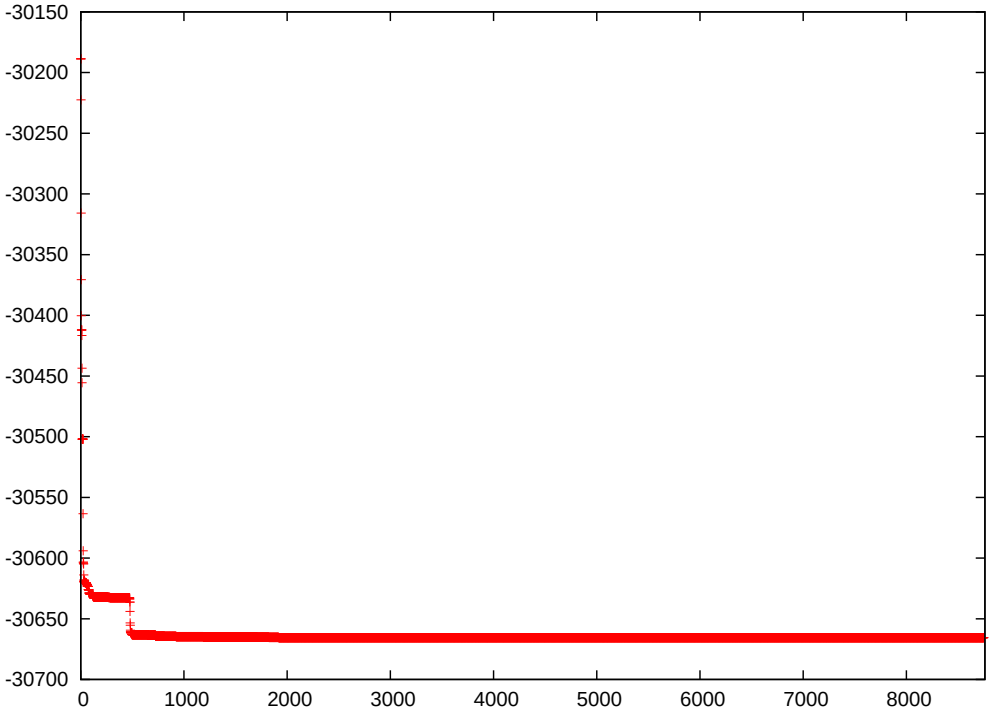

Figura A.4: Gráfica de convergencia para la función *g04*.

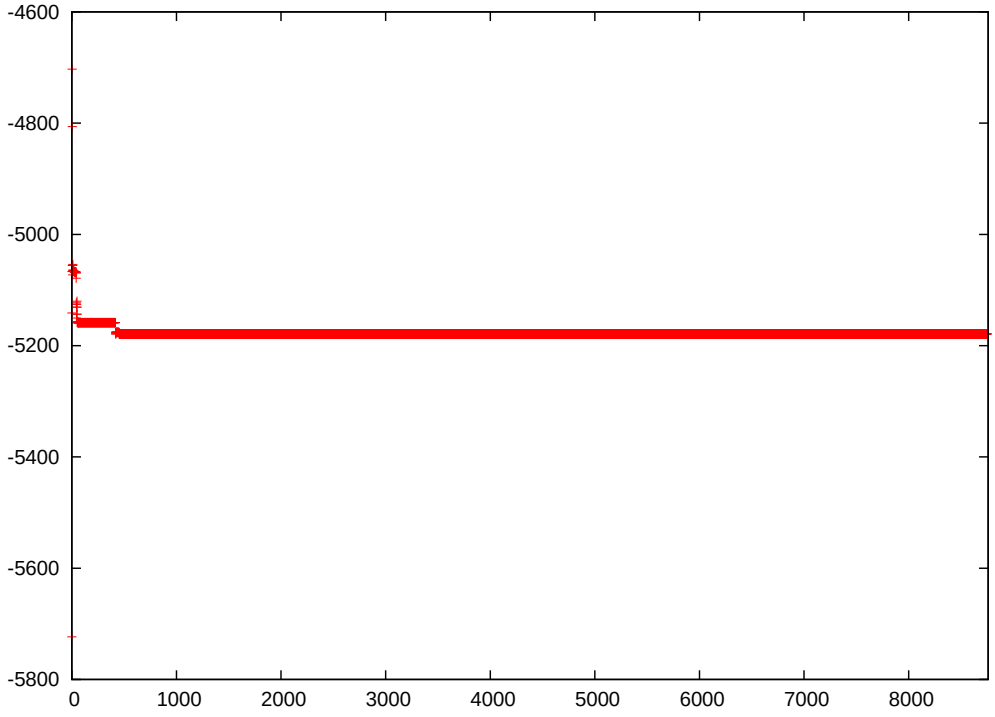

Figura A.5: Gráfica de convergencia para la función *g05*.

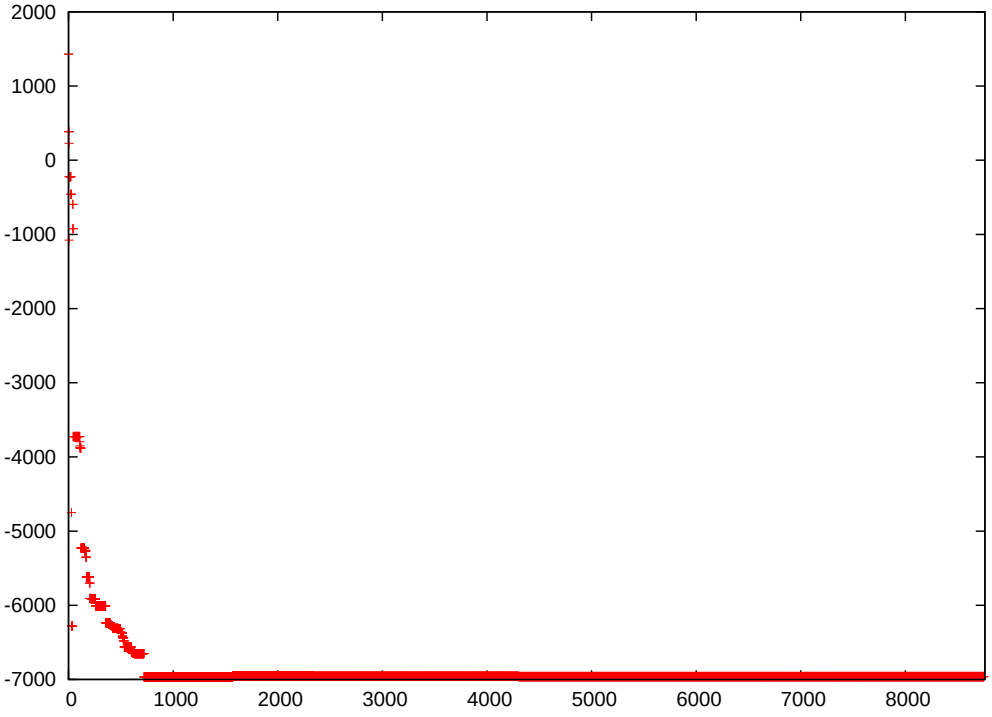

Figura A.6: Gráfica de convergencia para la función *g06*.

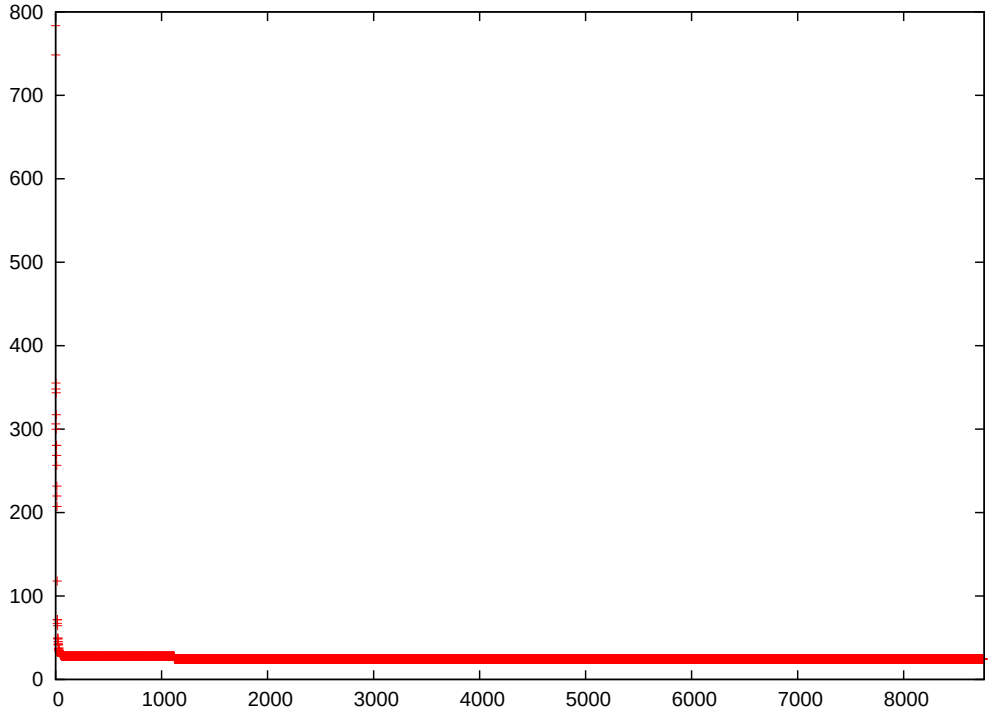

Figura A.7: Gráfica de convergencia para la función *g07*.

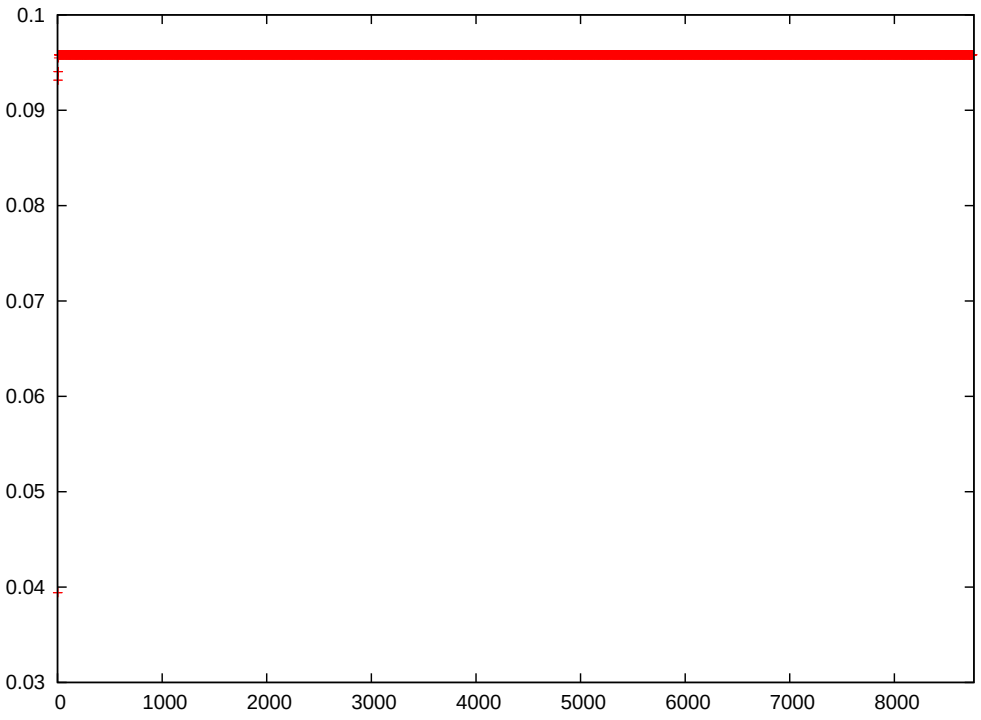

Figura A.8: Gráfica de convergencia para la función *g08*.

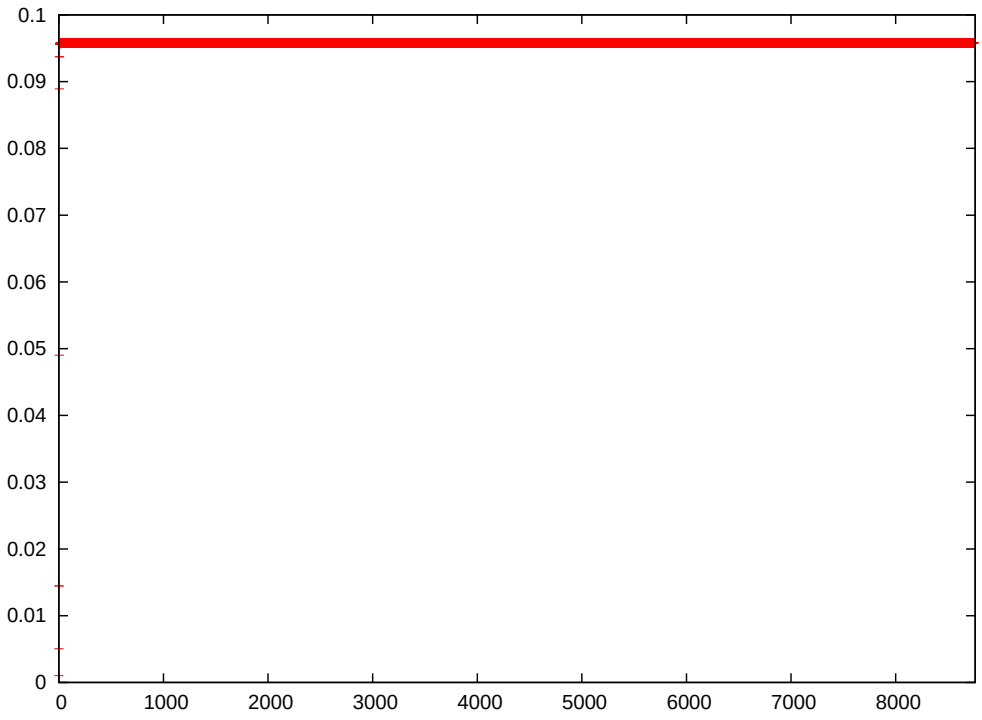

Figura A.9: Gráfica de convergencia para la función *g09*.

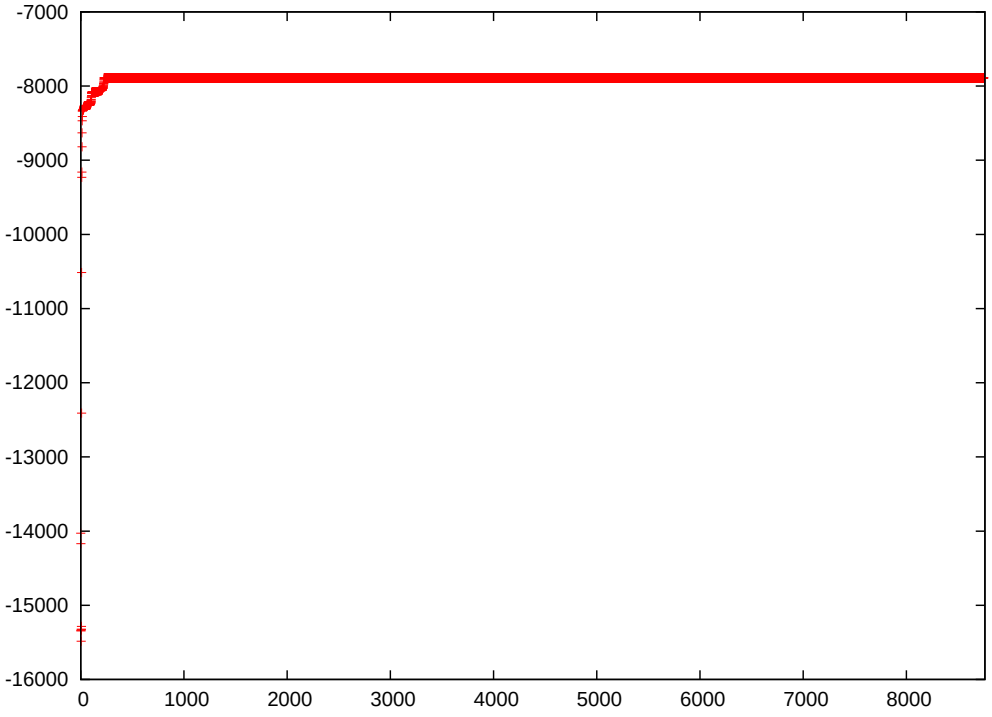

Figura A.10: Gráfica de convergencia para la función *g10*.

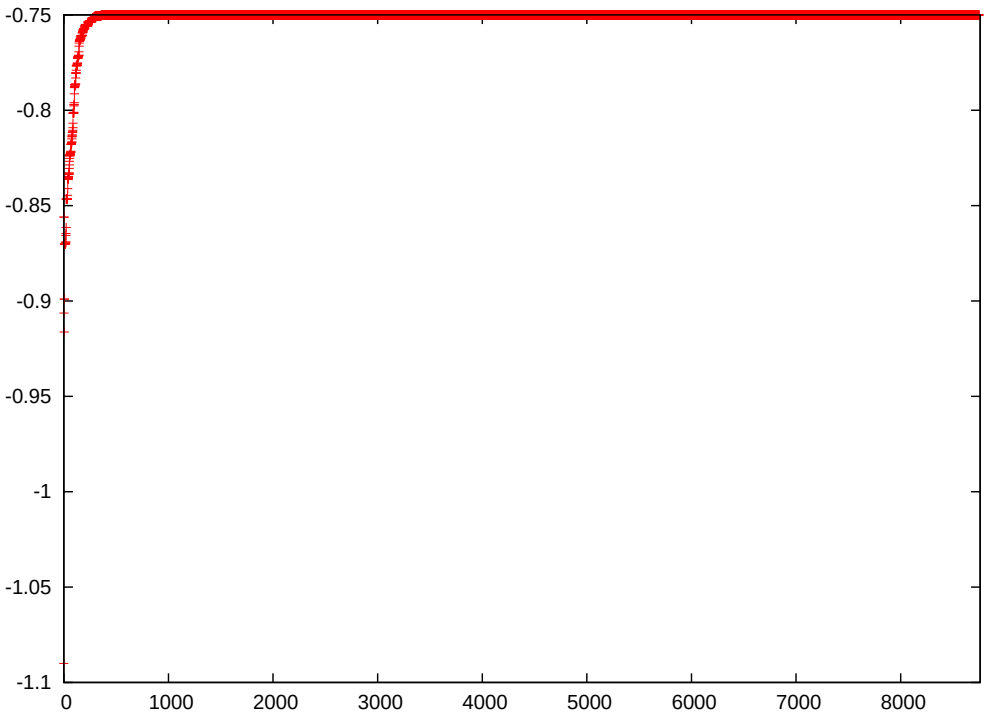

Figura A.11: Gráfica de convergencia para la función *g11*.

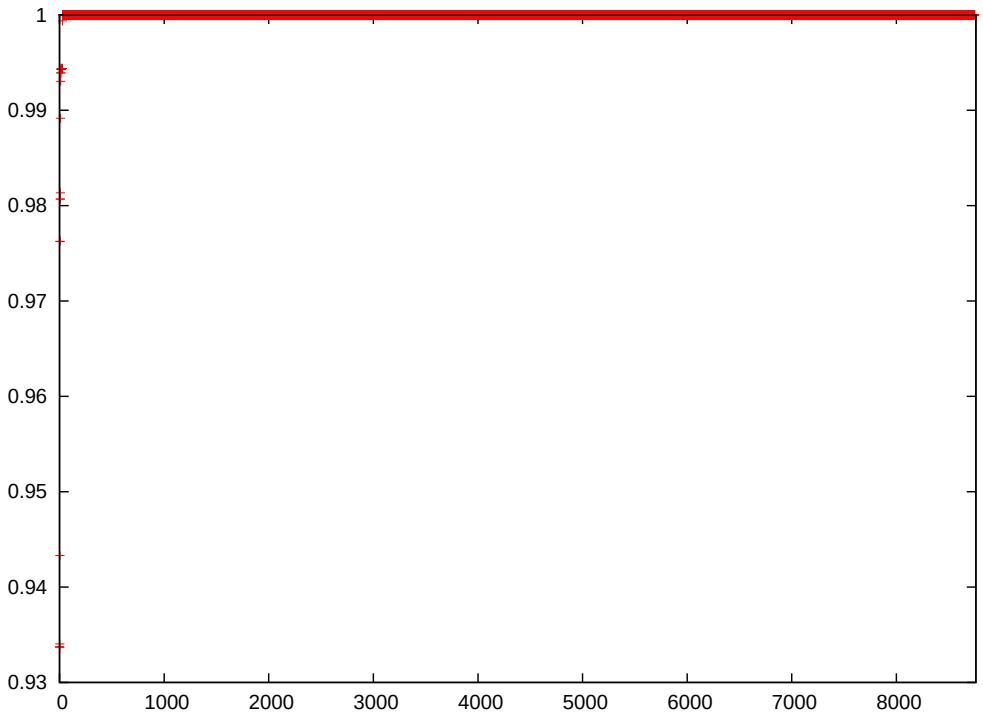

Figura A.12: Gráfica de convergencia para la función *g12*.

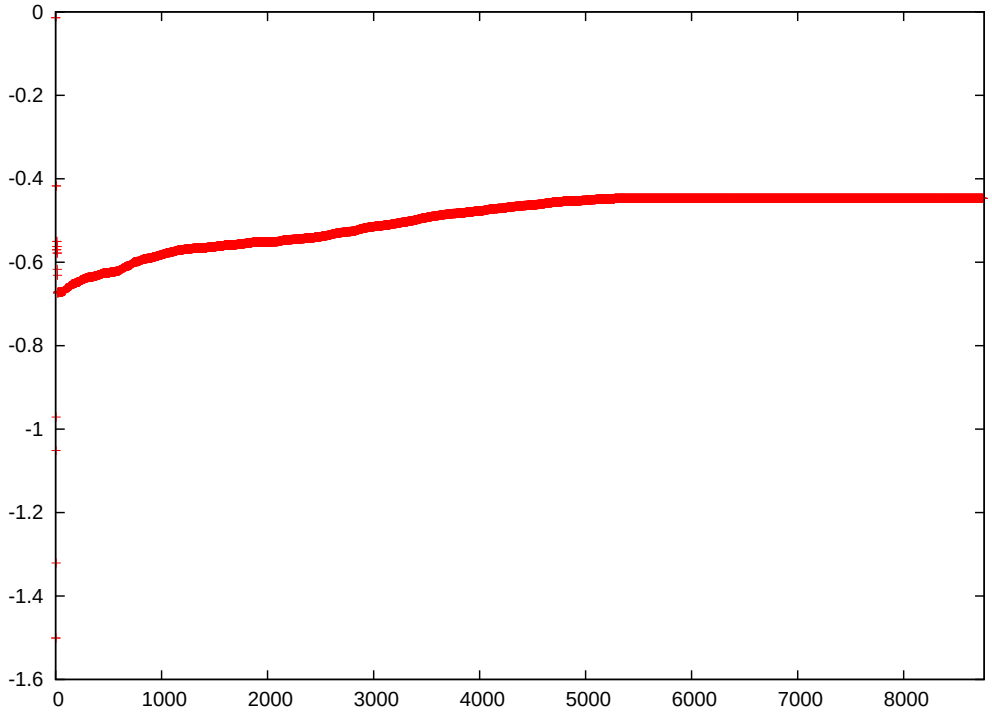

Figura A.13: Gráfica de convergencia para la función *g13*.

Apéndice B

En el presente apéndice se exponen los mejores resultados obtenidos con nuestra técnica denominada IPSO.

B.1. g01

El mejor valor encontrado por el algoritmo IPSO es: $f(x)$ =-15.00000, con $\vec{x}$ = (1.000000, 1.000000, 1.000000, 1.000000, 1.000000, 1.000000, 1.000000, 1.000000, 1.000000, 3.000000, 3.000000, 3.000000, 1.000000) y las restricciones con un valor de (-0.000000, -0.000000, -0.000000, -5.000000, -5.000000, -5.000000, -0.000000, -0.000000, -0.000000).

B.2. g02

El mejor valor encontrado por el algoritmo IPSO es: $f(x)$ =-0.803580, con $\vec{x}$ =(3.163669, 3.128001, 3.095444, 3.061935, 3.028568, 2.991514, 2.957399, 2.920100, 0.489789, 0.497012, 0.480988, 0.486680, 0.462717, 0.465487, 0.469523, 0.452925, 0.450834, 0.446648, 0.439984, 0.441946) y las restricciones con un valor de (-0.000000, -120.068837).

B.3. g03

El mejor valor encontrado por el algoritmo IPSO es: $f(x)$ =1.00044, con $\vec{x}$ =(0.315996, 0.316600, 0.314841, 0.317460, 0.316708, 0.315885, 0.316977, 0.316802, 0.315693, 0.315466) y las restricciones con un valor de (-0.000200, -0.000000).

B.4. g04

El mejor valor encontrado por el algoritmo IPSO es: $f(x)$ =-30665.537191, con $\vec{x}$ = (78.000000, 33.000000, 29.995264, 44.999997, 36.775794) y las restricciones con un valor de (-0.000003, -91.999997, -11.159503, -8.840497, -5.000000, -0.000000).

B.5. g05

El mejor valor encontrado por el algoritmo IPSO es: $f(x)$ =5126.50133, con $\vec{x}$ = (676.657563, 1029.576553, 0.121220, -0.395120) y las restricciones con un valor de (-0.000000, -0.000000, 0.001921, -0.033660, -1.066340, -0.002000, -0.002000, -0.003921).

B.6. g06

El mejor valor encontrado por el algoritmo IPSO es: $f(x)$ =-6961.85420, con $\vec{x}$ = (14.094984, 0.842925) y las restricciones con un valor de (-0.000016, 0.000047).

B.7. g07

El mejor valor encontrado por el algoritmo IPSO es: $f(x)$ =-6961.85420, con $\vec{x}$ =24.42530, con $\vec{x}$ = (2.218124, 2.288651, 8.711773, 5.131648, 0.964729, 1.406649, 1.389230, 9.872438, 8.358119, 8.311452) y las restricciones con un valor de (-0.000000, -0.000000, -0.000000, -1.964756, -0.000000, -0.000000, -6.042068, -49.563640).

B.8. g08

El mejor valor encontrado por el algoritmo IPSO es: $f(x)$ =-0.095825, con $\vec{x}$ = (1.227971, 4.245373) y las restricciones con un valor de (-1.737460, -0.167763).

B.9. g09

El mejor valor encontrado por el algoritmo IPSO es: $f(x)$ =680.65552, con $\vec{x}$ =(2.329891, 1.960406, -0.466128, 4.339778, -0.607185, 1.014050, 1.578092)

y las restricciones con un valor de (-0.000186, -252.689831, -145.024264, -0.000053).

B.10. g10

El mejor valor encontrado por el algoritmo IPSO es: $f(x)$ =7064.050416, con $\vec{x}$ =(701.608794, 1433.063370, 4929.378251, 191.418374, 302.824870, 208.581626, 288.593505, 402.824870) y las restricciones con un valor de (-0.000000, -0.000000, -0.000000, -0.000000, -0.000000, -0.000000).

B.11. g11

El mejor valor encontrado por el algoritmo IPSO es: $f(x)$ =0.749999, con $\vec{x}$ =(0.707375, 0.500380) y las restricciones con un valor de (-0.000000, -0.000101).

B.12. g12

El mejor valor encontrado por el algoritmo IPSO es: $f(x)$ =-1.000000, con $\vec{x}$ =(5.000000, 5.000000, 5.000000) y las restricciones con un valor de (-0.062500).

B.13. g13

El mejor valor encontrado por el algoritmo IPSO es: $f(x)$ =0.05489, con $\vec{x}$ = (-1.759748, 1.644769, 1.745934, -0.774881, -0.741213) y las restricciones con un valor de (-0.000200, -0.000000, -0.000000, -0.000000, -0.000200, -0.000200).

Bibliografía

[1] Thomas Bäck. Optimal mutation rates in genetic search. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 2–8, San Mateo, California, july 1993. Morgan Kaufmann.

[2] Thomas Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York, 1996.

[3] Thomas Bäck, Frank Hoffmeister, and Hans-Paul Schwefel. A survey of evolution strategies. In R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 2–9, San Mateo, California, 1997. Morgan Kaufmann Publishers.

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

[5] A. W. Burks. Computation, behavior and structure in fixed and growing automata. *Self-Organizing Systems*, pages 282–309, 1960.

[6] Carlos A. Coello Coello. An updated survey of evolutionary multiobjective optimization techniques : State of the art and future trends. In *1999 Congress on Evolutionary Computation*, pages 3–13, Washington, D.C., July 1999. IEEE Service Center.

[7] Carlos A. Coello Coello. Use of a self-adaptative penalty approach dor engineering optimization problems. *Computers in Industry*, 41(2):113–127, 2000.

[8] Paul R. Cohen. Empirical methods for artificial intelligence. The MIT Press, 1995.

[9] Kalyanmoy Deb. *Computer Methods in Applied Mechanics and Engineering*, chapter An Efficient Constraint Handling Method for Genetic Algorithms. In Press, 1999.

[10] Kalyanmoy Deb and David E. Goldberg. An investigation of niche and species formation in genetic in genetic function optimization. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 42–50, George Mason University, San Mateo, California, 1989. Morgan Kaufmann.

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

[12] Á. E. Eiben. *Introduction to evolutionary computing.* Springer, Berlin, Germany, 2003. ISBN: 3-540-40184-9.

[13] Andries P. Engelbrecht. *Computational intelligence.* Wiley, 2002.

[14] Andries P. Engelbrecht. *Computational Intelligence : An Introduction.* John Wiley & Sons, 2003. ISBN 0-4708-487-07.

[15] Larry J. Eshelman, editor. *Rudolphe G. Le Riche and Catherine Knopf-Lenoir and Raphael T. Haftka*, San Mateo, California, july 1995. University of Pittsburgh, Morgan Kaufmann Publishers.

[16] David B. Fogel. *Evolutionary Computation. Toward a New Philosophy of Machine Intelligence.* The Institute of Electrical and Elextronics Engineers, New York, 1995.

[17] Lawrence J. Fogel. *Artificial Intelligence through Simulated Evolution.* John Wiley, New York, 1966.

[18] Lawrence J. Fogel. *Artificial Intelligence through Simulated Evolution. Forty Years of Evolutionary Programming.* John Wiley and Sons, inc., 1999.

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

[20] John Holland. Concerning efficient adaptive systems. *Self-Organizing Sistems*, Spartan Books:215–230, 1962.

[21] John Holland. Outline for a logical theory of adaptative systems. *Journal of the Association for Computing Machinery*, 9:297–314, 1962.

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

[23] A. Homaifar, S. H. Y. Lai, and X.Qi. Constrined optimization via genetic algorithms. *Simulation*, 62(4):242–254, 1994.

[24] X. Hu and R. Eberhart. Solving constrained nonlinear optimization problems with particle swarm optimization. In *Proceedings of the 6th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2002)*, volume 5, Orlando, USA, IIIS., 2002.

[25] J. Joines and C. Houck. On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with gas. In David Fogel, editor, *Proceedings of the firts IEEE Conference on Evolutionary Computation*, pages 579–584, Orlando, Florida, 1994. IEEE Press.

[26] 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.

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

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

[29] S. Koziel and Z. Michalewicz. Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization. *Evol. Comput.*, 7(1):19–44, 1999.

[30] Zbigniew Michalewicz. A survey of constraint handling techniques in evolutionary computation methods. In J. R. McDonnell, R. G. Reynolds, and D.B. Fogel, editors, *Proceedings of the 4th Annual Conference on Evolutionary Programming*, pages 135–155, Cambridge, Massachusetts, 1995. the MIT Press.

[31] Zbigniew Michalewicz and Naguib F. Attia. Evolutionary optimization of constrained problems. In *Proceedings of the 3rd Annual Conference on Evolutionary Programming*, pages 98–108. Word Scientific, 1994.

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

[33] K. Parsopoulos and M. Vrahatis. Particle swarm optimization method for constrined optimization problems. In P. Sincak, J. Vascak, V. Kvasnicka, and J. Pospicha, editors, *Intelligent Technologies*, volume 76 of *Frontiers in Artificial Intelligence and Applications series*, pages 214–220, 2002. ISBN: 1-58603-256-9.

[34] Gregorio Toscano Pulido and Carlos A. Coello Coello. A constraint-handling mechanism for particle swarm optimization. *Proceedings of the 2004 Congress on Evolutionary Computation, IEEE*, pages 1396–1403, 2004.

[35] Jon T. Richardson, Mark R. Palmer, Gunar Liepins, and Mike Hilliard. Some guidelines for the genetics algorithms with penalty functions. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 191–197. Morgan Kaufmann Publishers, 1989.

[36] Tomas P. Runarsson and Xin Yao. Stochastic ranking for constrained evolutionary optimization. *IEEE Transactions on Evolutionary Computation*, 4(3), September 2000.

[37] Hans-Paul Schwefel. *Numerical Optimization of Computer Models*. John Wiley and Sons, Great Britain, 1981.

[38] O. G. Selfridge. Pandemonium: A paradigm for learning. *Proceedings of the Symposium on Mechanization of Thought Processes*, pages 511–529, 1958.