Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Departamento de Ingeniería Eléctrica

Sección de Computación

Propuesta de Evolución Diferencial para Optimización de Espacios Restringidos

Tesis que presenta

Jesús Velázquez Reyes

para obtener el Grado de

Maestro en Ciencias

en la Especialidad de

Ingeniería Eléctrica

Director de la Tesis

Dr. Carlos Artemio Coello Coello

México, D.F. **Enero 2006**

Una vez hecho algo, no puede
valer mucho; es una obra
humana con todas las
imperfecciones de lo humano,
pero el hecho de ejecutarla
sí es interesante.

No sé si la instrucción puede
salvarnos, pero no sé de nada
mejor.

Que otros se jacten de las
páginas que han escrito; a mi
me enorgullecen las que he
leído.

Publicamos nuestro libros
para librarnos de ellos, para
no pasar el resto de nuestras
vidas corrigiendo borradores.

Jorge Luis Borges (1899-1986)

A Dios, al ser que abarca más allá del tiempo, la materia, la ciencia y la religión; a ese ser que compone a todas las cosas y del cual formamos parte. Gracias por este instante.

A mi Mamá y a mi Papá, porque gracias a su amor y esfuerzo incondicional he podido ser lo que soy, siempre estaré en deuda con ustedes.

A Abraham y Daniel, porque en su vivir moldearon mi forma de ser.

A Nancy Noemí, porque uno está enamorado cuando se da cuenta de que la otra persona es única. *E-dé, e-mé* [1]

Al güero, porque me trajo una felicidad inesperada.

[1] *Instinto de Inez, Carlos Fuentes, Editorial Alfaguara, página 63*

Agradecimientos

Agradezco a Dios por darme la oportunidad de vivir y terminar este ciclo.

A mis padres que con su esfuerzo, sacrificio, dedicación y sobretodo amor han forjado mi ser.

A mis hermanos Daniel y Abraham que me han apoyado y que sin darse cuenta han colaborado enormemente en lo que soy.

A Nancy Noemí que estuvo junto a mi durante este largo camino dándome apoyo y soporte en todo momento, y que con su amor me hizo vivir momentos que atesoraré siempre.

A mis muy queridos amigos Gibran y Luis Alfonso por estos nueve años de amistad que hemos vivido y por el largo camino que hemos recorrido.

A mis no menos queridos amigos Jonathan, Daniel, Rogelio, Emmanuel, Wilhem, Sabel, Marisol y Flavio que han compartido su amistad conmigo.

Al güero que trajo nuevas alegrías a nuestra familia y que se ha convertido en un miembro más de ella.

Al Dr. Carlos A. Coello Coello por su enorme e incondicional apoyo durante mi estancia en el CINVESTAV, gracias por todo el tiempo y apoyo ofrecidos.

A mis sinodales, Dr. Arturo Díaz Pérez y Dr. Francisco José Rodríguez Henríquez, por revisar mi tesis, permitiéndome a través de sus valiosas correcciones obtener un mejor trabajo.

A Sofi por su apoyo y cariño para con nosotros que sobrepasa toda obligación laboral; en verdad tu cariño es correspondido.

A mi *Alma mater* el Instituto Politécnico Nacional que fue pieza fundamental en mi formación durante estos nueve años de estudio, gracias por ofrecerme siempre lo mejor.

Al CINVESTAV por concederme la oportunidad de pertenecer a esta gran casa de estudios, además de todos los apoyos recibidos.

de estudiantes y contribuir en mi educación a lo largo de estos dos años.

Al CONACyT por sustentar económicamente mis estudios de maestría, permitiéndome así culminar esta meta.

Agradezco la beca terminal de maestría otorgada por CONACyT a través del proyecto de investigación "Técnicas Avanzadas de Optimización Evolutiva Multiobjetivo"(Ref. 45683-Y), cuyo responsable es el Dr. Carlos A. Coello Coello.

Al personal administrativo de servicios escolares y de la biblioteca de Ingeniería eléctrica por las facilidades otorgadas en cada trámite y consultas bibliográficas que realicé.

Resumen [1]

En los problemas de optimización del mundo real existen restricciones de diferentes tipos (p. ej. físicas, de tiempo, geométricas, etc.) las cuales modifican la forma del espacio de búsqueda. Durante los últimos años, una variedad de meta-heurísticas han sido diseñadas y aplicadas para la solución de problemas de optimización restringida donde no hay métodos clásicos de programación matemática disponibles.

Los Algoritmos Evolutivos y otras meta-heurísticas operan como técnicas de búsqueda sin restricciones cuando son usados en optimización. Así, éstas requieren un mecanismo adicional que incorpore las restricciones del problema en su estructura.

Históricamente, en los algoritmos evolutivos y en la programación matemática la propuesta más común de incorporación de restricciones es usando funciones de penalización, las cuales fueron originalmente propuestas en la década de los 40s y después fueron expandidas por muchos investigadores. En general, las funciones de penalización tienen varias limitaciones. Una de las más importantes es la necesidad de ajustar los factores de penalización. En algunos casos, existen muchos factores que deben ser ajustados y en la mayoría de los casos sus valores son dependientes del problema.

La Evolución Diferencial es un método estocástico global de búsqueda directa, el cual forma parte de estas nuevas meta-heurísticas y que ha probado ser un procedimiento muy efectivo en la optimización numérica sin restricciones. Así, ha habido algunas propuestas para extender el algoritmo de Evolución Diferencial a espacios de búsqueda restringidos.

En esta tesis se propone una nueva versión del algoritmo de Evolución Diferencial para resolver problemas de optimización restringida. Se propone un nuevo modelo así como un mecanismo de manejo de restricciones basado en funciones de penalización, en el cual no es necesario ajustar factores de penalización.

Esta nueva versión del algoritmo de Evolución Diferencial es comparada contra los algoritmos más representativos en optimización restringida usando un conjunto de problemas de prueba estándar utilizados en el área de Computación Evolutiva.

[1] Documento elaborado en LaTeX2e

Abstract

In real-world optimization problems there are constraints of different types (e.g. physical, time, geometric, etc.) which modify the shape of the search space. During the last few years, a wide variety of metaheuristics have been designed and applied to solve constrained optimization problems where classical mathematical programming methods are not available.

Evolutionary algorithms and other metaheuristics operate as unconstrained search techniques, when they are used for optimization. Therefore, they require an additional mechanism to incorporate constraints into their structure.

Historically, in both evolutionary algorithms and mathematical programming, the most common approach to incorporate constraints is the usage of penalty functions, which were originally proposed in the 1940s and later expanded by many researchers. In general, penalty functions have several limitations. One of their most important limitations is the problem of fine-tuning the penalty factors. In some cases, there are too many factors to be adjusted and in most cases they are problem-dependent.

Differential Evolution is a stochastic global direct search method. It is part of the aforementioned new metaheuristics and it has been found to be a very effective unconstrained numerical optimization procedure. So, there have been a few attempts to extend the Differential Evolution algorithm to constrained search spaces.

In this thesis we propose a new version of the Differential Evolution algorithm to solve constrained optimization problems. We propose a new model and also a constraint-handling mechanism based on penalty functions, in which there is no need of fine-tuning the penalty factors.

This new version of the Differential Evolution algorithm is compared against the most representative algorithms in constrained optimization using a well-known benchmark commonly adopted in Evolutionary Computation.

Índice general

Índice de figuras

Índice de tablas

Lista de Algoritmos

Capítulo 1

Introducción

La optimización es el acto de obtener el mejor resultado bajo ciertas circunstancias. Los ingenieros toman muchas decisiones de naturaleza tecnológica o administrativa en cualquier sistema de ingeniería. Esto con la finalidad de reducir el esfuerzo requerido o maximizar un beneficio deseado [76].

Así, la *optimización* puede ser definida como *el proceso de encontrar la mejor solución posible a un problema, bajo ciertas circunstancias.*

No existe un solo método capaz de resolver todos los problemas de optimización de manera eficiente bajo cualquier circunstancia [76]. Por lo tanto, se han desarrollado diferentes métodos para resolver clases específicas de problemas.

En el mundo real se presenta una amplia gama de problemas de optimización, entre los cuales se incluyen: optimización estructural, diseño en ingeniería, diseño de circuitos VLSI, diseño mecánico, nuclear, químico, y de control entre otros [76]. Aunque se han desarrollado técnicas de programación matemática para resolver problemas con propiedades particulares (p. ej. programación lineal, programación cuadrática, etc.) el problema general de la programación no lineal (con o sin restricciones) sigue siendo abierto y, dada su complejidad han surgido métodos alternativos para la solución de estos problemas. Una clase importante de estos métodos hoy en día son los Algoritmos Evolutivos.

Los Algoritmos Evolutivos (AE) han demostrado ser técnicas competitivas para la solución de varios problemas con características que los hacen difíciles de resolver por técnicas clásicas de programación matemática. Dentro de éstas están: la no linealidad, la alta modalidad, el ruido, la imposibilidad de calcular derivadas de cualquier orden, la no continuidad, entre otras. En particular, una nueva técnica en el área de AE llamada Evolución Diferencial

[85] ha mostrado ser competitiva en la optimización de problemas con tales características.

La Evolución Diferencial (ED) es un algoritmo simple de búsqueda directa empleado en optimización numérica global, la cual fue propuesta por Price y Storn [85], y ésta forma parte de los Algoritmos llamados Evolutivos. Lo que hace diferente al algoritmo de Evolución Diferencial de otros Algoritmos Evolutivos es la forma en que emplea combinaciones lineales de individuos en la generación nuevas soluciones que "heredan" características deseables para la solución del problema.

En su forma original no maneja espacios de búsqueda restringidos [85]. Se han propuesto diferentes mecanismos para el manejo de restricciones principalmente basados en funciones de penalización [75]. Estas presentan una serie de desventajas siendo la más importante el ajuste de los coeficientes de penalización, ya que éstos son dependientes del problema [6].

Existen diferentes modelos de la Evolución Diferencial reportados en la literatura [75, 85]. Vitaliy Feoktistov y Stefan Janaqi [27] presentan nuevos modelos de la Evolución Diferencial que pretenden mejorar el rendimiento de la técnica imitando a las técnicas clásicas de descenso empinado (bajando la colina). Sin embargo, estas nuevas propuestas no fueron probadas usando funciones de prueba estándar en la literatura de algoritmos evolutivos para optimización numérica sin restricciones [92].

Por tal motivo, en esta tesis se presenta un estudio comparativo entre los principales modelos de Evolución Diferencial empleando en ello un conjunto de funciones de prueba estándar en la literatura para optimización global sin restricciones. Los resultados de este estudio se muestran en el capítulo 4.

El objetivo principal de este trabajo de tesis es **explorar** las posibilidades de la Evolución Diferencial como optimizador global en **espacios restringidos** empleando diferentes modelos del algoritmo; así, como otro mecanismo de manejo de restricciones. Para ello se propone un nuevo modelo de recombinación a la Evolución Diferencial que mejora la velocidad de convergencia sin sacrificar su capacidad de búsqueda, este nuevo modelo se compara con los modelos de ED que muestran los mejores rendimientos (en términos de la calidad de las soluciones generadas), los cuales están presentes en la literatura. Además, se propone un nuevo mecanismo de manejo de restricciones que basa su funcionamiento en una función de penalización exterior donde no es necesario ajustar los factores de penalización, este nuevo mecanismo evita que se emplee únicamente como criterio de comparación a la función objetivo o a las restricciones del problema exclusivamente.

El algoritmo propuesto es comparado contra algoritmos evolutivos competitivos empleados para la optimización numérica con restricciones. Se usa un conjunto de problemas de

prueba estándar en la literatura [80] para determinar el rendimiento de la propuesta. Con el fin de comparar los diferentes algoritmos, se proponen tres diferentes formas de comparación, las cuales no suponen normalidad en las muestras. Además, no se limitan a comparar conjuntos finales de soluciones, sino que determinan el comportamiento promedio del algoritmo a lo largo del proceso evolutivo.

Así, las aportaciones de esta tesis son:

- Un estudio comparativo entre los diferentes modelos de ED existentes en la literatura, para ello se emplea un conjunto de funciones de prueba estándar en la literatura de algoritmos evolutivos.
- Se propone un nuevo modelo de ED que mejora el desempeño del algoritmo. Este nuevo modelo se compara contra los mejores modelos existentes en la literatura empleando el conjunto de funciones de prueba, mostrando mejores resultados.
- Se propone un nuevo mecanismo de manejo de restricciones que junto con el nuevo modelo de ED muestra resultados competitivos en comparación con algoritmos representativos en el área de optimización numérica con restricciones en los algoritmos evolutivos.
- Se proponen tres formas de comparación de los algoritmos, las cuales no suponen normalidad en las muestras, y además determinan el comportamiento promedio del algoritmo a lo largo del proceso evolutivo.

1.1. Organización del Documento

Capítulo 1. Introducción: Este capítulo.

Capítulo 2. Introducción a la Computación Evolutiva: Se da al lector, una introducción al área de la computación evolutiva, además de describir la motivación del área así como sus principales componentes y paradigmas.

Capítulo 3. Nociones de Optimización: Se presentan las nociones básicas de optimización y se establece el planteamiento del problema que atañe a esta tesis. Se listan las características de los problemas de optimización restringida y se muestran las principales diferencias con la optimización sin restricciones.

Capítulo 4. Evolución Diferencial: Se presenta una descripción completa del algoritmo de la Evolución Diferencial mostrando las principales características de cada uno de sus componentes.

Capítulo 5. Algoritmos Evolutivos para Optimización Restringida: Se presenta una introducción de las técnicas más empleadas en los algoritmos evolutivos para manejo de restricciones.

Capítulo 6. Descripción de la Propuesta: Se presenta la descripción completa de la propuesta del algoritmo de Evolución Diferencial la cual consiste en un nuevo modelo de evolución diferencial y un mecanismo de manejo de restricciones.

Capítulo 7. Resultados Experimentales: Se muestran los resultados de un estudio comparativo del nuevo modelo de Evolución Diferencial contra otros modelos tradicionales empleando un conjunto de funciones de prueba estándar utilizadas en la optimización global sin restricciones. En la segunda parte se muestra una comparativa del nuevo modelo de Evolución Diferencial junto con el mecanismo propuesto de manejo de restricciones contra los algoritmos más representativos del área de optimización en espacios restringidos de la computación evolutiva.

Capítulo 8. Conclusiones y Trabajo Futuro: Se muestra las conclusiones basadas en los resultados obtenidos, así como las posibles lineas de investigación a seguir.

Apéndice A: Funciones Sin Restricciones: Se listan las descripciones de las funciones escalables sin restricciones empleadas en la comparación de los diferentes modelos de Evolución Diferencial.

Apéndice B: Funciones Con Restricciones: Se listan los trece problemas de prueba estándar empleados en la optimización numérica con restricciones en el área de Computación Evolutiva. Estos problemas son empleados en la comparación de la propuesta contra los algoritmos representativos del área.

Apéndice C: Gráficas de Resultados: Para el algoritmo propuesto se muestran los histogramas de frecuencia de las soluciones, las distribuciones de la distancia promedio al valor óptimo y las gráficas de la distancia al valor óptimo con respecto al número de evaluaciones de la función objetivo para cada uno de los trece problemas de prueba con restricciones.

Capítulo 2

Introducción a la Computación Evolutiva

En este capítulo se da una introducción al lector al área de la computación evolutiva. Se describe la motivación del área así como sus principales componentes y paradigmas. Finalmente, se reseñan los desarrollos de nuevas heurísticas diseñadas para dominios específicos. Estas nuevas heurísticas están basadas en la simulación de una variedad de comportamientos biológicos y no sólo en la evolución natural. Por tal motivo, tales heurísticas son denominadas bio-inspiradas.

2.1. Introducción

La Computación Evolutiva (CE) es un área de las ciencias de la computación; como su nombre lo sugiere, es un tipo de computación inspirada en los procesos de la evolución natural.

El hecho de que algunos científicos hayan elegido a la evolución natural como una fuente de inspiración no es sorprendente. El poder de la evolución natural es evidente en la diversidad de las especies que conforman nuestro mundo, cada una de ellas adaptada a su propio ecosistema con el fin de sobrevivir.

2.2. Neodarwinismo

La teoría evolutiva universalmente aceptada hoy en día es el *Neodarwinismo*, la cual está basada en la teoría evolutiva clásica de Charles Darwin, la teoría selectiva de Weis-

mann y las leyes de la genética de Mendel.

El Neodarwinismo establece que la diversidad actual de seres vivos es consecuencia de unos cuantos procesos estocásticos que actúan en las especies y sus poblaciones. Estos procesos son: *reproducción*, *mutación*, *competencia* y *selección*.

La **reproducción** es una propiedad obvia de las especies *existentes*. Toda especie que pierda su capacidad reproductiva está condenada a la extinción. La reproducción se logra a través de la transferencia de la información genética (sexual o asexual) de los individuos a sus descendientes.

La **mutación** es una alteración de la información genética en el momento que ésta es transferida durante la reproducción.

La **competencia** es la consecuencia del crecimiento de las poblaciones de especies en un espacio con recursos finitos. La **selección** es el resultado de la competencia entre individuos y especies por ocupar el espacio y los recursos disponibles.

Así, la **evolución** surge como un resultado **inevitable** de la interacción de estos procesos físicos estocásticos básicos.

Los individuos y las especies pueden ser vistos como una dualidad de su información genética (**genotipo**) y sus características físicas y de comportamiento (**fenotipo**). El *genotipo* provee un medio de almacenamiento a la información históricamente adquirida. La relación entre el fenotipo y el genotipo es compleja y no lineal. Dos fenómenos determinan esta relación: la *pleitropía* y la *poligenia*.

La **pleitropía** es el efecto donde un segmento de la información genética simultáneamente afecta a varias características en el fenotipo y la **poligenia** es el efecto donde una característica fenotípica es determinada por la interacción simultánea de varios segmentos de información genética. Así, el fenotipo varía de acuerdo a la interacción entre la estructura genética y las condiciones ambientales presentes.

El Neodarwinismo establece que la selección natural es la fuerza predominante que conduce las características fenotípicas de los organismos en la mayoría de las situaciones. Los cambios en el fenotipo son consecuencia de la mutación, la recombinación y las restricciones del ambiente; pero, la selección es quien dirige tales cambios.

La selección preserva o incrementa la *aptitud* de las poblaciones de individuos, donde **aptitud** se define como la habilidad de sobrevivir y reproducirse en un ambiente específico.

La selección únicamente actúa sobre los individuos y las especies.

El concepto de topología adaptativa ayuda a describir la aptitud de las especies y sus individuos. Todos los fenotipos posibles se localizan en una gráfica (a la que se denomina paisaje de aptitud) de acuerdo a su valor de aptitud. El proceso evolutivo selecciona los fenotipos que más rápidamente se adapten a regiones de mayor aptitud.

En la figura 2.1 se muestra un paisaje de aptitud hipotético, donde los fenotipos avanzan hacia mejores regiones como consecuencia de la variación genética y la selección natural. Una población se puede representar como un conjunto de puntos en la gráfica, donde cada punto representa un fenotipo de la población.

En la evolución natural el paisaje de aptitud es dinámico, de tal manera que los fenotipos continuamente evolucionan hacia regiones de mejor aptitud y evitando el estancamiento.

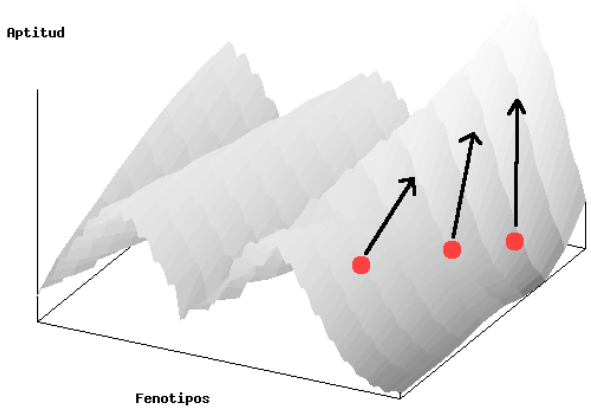


Figura 2.1: Paisaje de aptitud hipotético

Visto de esta manera, la evolución se observa como un proceso de solución a problemas dadas ciertas condiciones iniciales y restricciones del ambiente.

El ambiente está en constante cambio; así, las especies están en una constante evolución adaptándose a los cambios en el paisaje de aptitud y ningún organismo se puede considerar

como perfectamente adaptado a su ambiente.

En [28] se presenta un resumen de las principales características del *Neodarwinismo*, las cuales se listan a continuación:

1. El individuo es el principal objeto de la *selección*.
2. Las variaciones genéticas son principalmente fenómenos aleatorios. Los procesos estocásticos tiene un papel muy importante en la evolución.
3. Las variaciones *genotípicas* son principalmente causadas por la *recombinación* y en última instancia por la *mutación*.
4. La evolución *gradual* puede ocasionar discontinuidades fenotípicas.
5. No todos los cambios fenotípicos son consecuencia de la selección natural.
6. La selección es probabilística, no determinista.

Cualquier simulación computacional que se realice de la evolución debería basarse en estos principios.

2.3. Paradigma de la Computación Evolutiva

La *evolución* es intrínsecamente un método de búsqueda. La flora y la fauna de nuestro planeta demuestran un comportamiento complejo resultado de un proceso de optimización en todos los niveles: célula, órgano, individuo, población, especie.

Los problemas que las especies biológicas han resuelto son caracterizados por el caos, el azar, la temporalidad y la no linealidad. Éstas son a su vez las características de los problemas que han demostrado ser especialmente intratables por métodos clásicos de optimización.

Así, la metáfora fundamental de la computación evolutiva es considerar a la evolución un **método robusto de búsqueda**, donde se trata de aplicar los procesos evolutivos (*reproducción, mutación, competencia y selección*) a la solución de problemas donde no hay métodos disponibles o donde las soluciones son insatisfactorias.

El concepto de *método robusto de búsqueda* se muestra en la figura 2.2, donde se ilustra el hecho de que existen métodos especializados y eficientes (en términos de la calidad de resultados con respecto a un conjunto de recursos dado) para la solución de ciertas clases

de problemas; pero, hay algunos tipos de problemas donde no existen métodos que los resuelvan de manera satisfactoria.

Así, un *método robusto de búsqueda* proporciona una opción de solución para un rango amplio de problemas donde no hay métodos especializados o las soluciones son insatisfactorias. Cabe mencionar que en ningún caso (métodos especializados o esquemas robustos) existe un algoritmo que resuelva todas las clases de problemas.

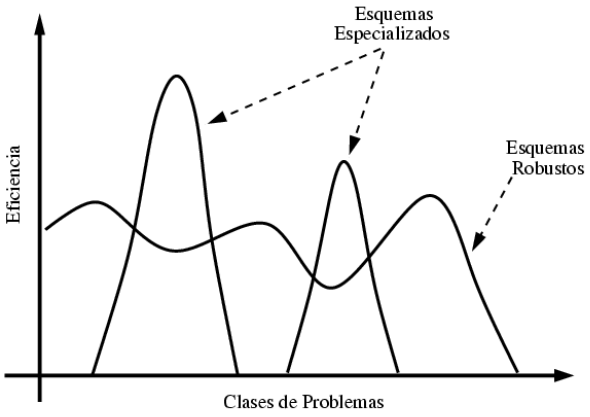


Figura 2.2: Tipos de problemas y esquemas

Los algoritmos evolutivos son métodos robustos de búsqueda inspirados en los procesos de la evolución natural. Estos algoritmos proporcionan una opción a la solución de problemas intratables por métodos clásicos. Los algoritmos evolutivos no garantizan optimalidad, sino que ofrecen un buen compromiso entre los recursos empleados para la solución del problema y la calidad de la solución obtenida.

2.4. Algoritmos Evolutivos

Hoy día existe una extensa colección de algoritmos evolutivos empleados para la solución de una amplia gama de problemas. En la mayoría de ellos subyace una base común.

Dada una función de *calidad* a ser optimizada (minimización o maximización) se genera un agregado de soluciones *candidatas* (v.g. elementos pertenecientes al dominio de la función de calidad). A cada elemento se asigna una medida (**aptitud**) basada en el valor que

toma la función de calidad dada la solución *candidata*.

Con base en sus valores de aptitud algunas soluciones candidatas son elegidas como semillas de nuevas soluciones. Las nuevas soluciones (*descendientes*) se generan aplicando los procesos de recombinación y mutación a las soluciones seleccionadas.

En la mayoría de los casos las soluciones candidatas y sus descendientes compiten por un lugar en la población de la siguiente generación. La competencia es basada principalmente en el valor de aptitud de las soluciones pero, también pueden influir otros criterios (p. ej. la *edad*, la dispersión, etc).

Este proceso es iterado hasta que se obtiene una solución candidata adecuada o apropiada para el problema, o los límites de los recursos computacionales previamente establecidos han sido alcanzados.

```
1 Inicializar(P_g = {x_1, x_2, ··· , x_np});
2 Evaluar(P_g) {f(x_1), f(x_2), ··· , f(x_np)} ;
3 while Criterio de Terminación NO satisfecho do
4 |   padres ← SeleccionarPadres(P_g) ;
5 |   P'_g ← Recombinar(padres) ;
6 |   P''_g ← Mutar(P'_g) ;
7 |   Evaluar(P''_g) {f(x''_1), f(x''_2), ··· , f(x''_np)} ;
8 |   P_{g+1} ← Seleccionar(P''_g ∪ P_g) ;
9 |   G ← G + 1 ;
```

Algoritmo 1: Esquema general de un algoritmo evolutivo

En el algoritmo 1 se muestra una descripción en pseudocódigo de un algoritmo evolutivo, donde $P_g = \{\vec{x}_1, \vec{x}_2, \cdots, \vec{x}_{np}\}$ representan a un conjunto de soluciones *candidatas* (la población actual). La función $EvaluarP''_g = \{f(\vec{x}''_1), f(\vec{x}''_2), \cdots, f(\vec{x}''_{np})\}$ asigna a cada solución un valor de *aptitud* y las funciones $seleccionar, recombinar, mutar$ representan los operadores de selección, recombinación y mutación respectivamente.

De la descripción anterior se observa que este esquema pertenece a la categoría de algoritmos *generar-y-probar*. La evaluación de la función de calidad representa una estimación heurística de la condición de las soluciones *candidatas* (valor de aptitud), mientras el proceso de búsqueda se basa en la variación de las soluciones resultado de los operadores de recombinación y mutación.

Por último, es importante mencionar que los principales elementos que componen a un algoritmo evolutivo son estocásticos. Durante el proceso de selección los individuos más aptos tienen mayor posibilidad de convertirse en *semillas* de los descendientes; sin embargo, los menos aptos también tienen posibilidades. En la recombinación, los segmentos de información transferidos a los descendientes son seleccionados de manera aleatoria; mientras que, la mutación es un proceso intrínsecamente estocástico donde se generan perturbaciones aleatorias a los individuos.

2.4.1. Principales Componentes de los Algoritmos Evolutivos

En cada uno de los diferentes tipos de algoritmos evolutivos se especifica un número de componentes, procedimientos y operadores que los definen.

Los principales componentes son los siguientes:

1. Definición de los individuos (representación)
2. Función de evaluación (función de calidad)
3. Mecanismo de selección de *padres*
4. Operadores de recombinación y mutación
5. Mecanismo de supervivencia

Definición de los Individuos

El primer paso para definir un algoritmo evolutivo es representar el contexto del problema original al espacio solución, donde la evolución toma lugar. Los objetos que conforman posibles soluciones en el contexto del problema original son llamados fenotipos; mientras que, la codificación del fenotipo es llamada genotipo. Durante la definición de la estructura de los individuos es necesario establecer una relación entre el fenotipo y el genotipo.

Cabe mencionar que los operadores de recombinación y mutación generalmente obran en el espacio genotípico mientras que la selección toma lugar en el espacio fenotípico.

Los espacios fenotípicos y genotípicos son de naturalezas diferentes, pero el mapeo del genotipo al fenotipo debe ser único. En la terminología común se suele llamar de manera indistinta a los puntos del espacio de búsqueda en el contexto original del problema como: *soluciones candidatas, fenotipos o individuos.*

Función de Evaluación

La función de evaluación representan los requerimientos a los cuales las soluciones se adaptan. Desde la perspectiva del problema, esta función representa la tarea a resolver por el algoritmo evolutivo. Técnicamente es una función o procedimiento que asigna una medida de *calidad* a los fenotipos dependiendo del grado al cual resuelven el problema.

En la terminología del área es comúnmente llamada función de **aptitud**.

Mecanismo de Selección de los *padres*

Durante la selección de los *padres*, también llamado proceso de *selección de cruce*, se elige a los individuos que serán las semillas para los descendientes; tal elección es basada en los valores de aptitud de los individuos y típicamente es probabilística.

Operadores de Recombinación y Mutación

La tarea de los operadores de *variación* (mutación y recombinación) es crear nuevos individuos a partir de los individuos actuales.

La **mutación** es un operador que siempre es estocástico. En general, la mutación produce cambios aleatorios sin sesgo; el propósito de este operador varía de acuerdo al tipo de algoritmo evolutivo empleado. Por ejemplo, en las *estrategias evolutivas* la mutación es el principal operador de variación y su papel es perturbar de manera acotada a las soluciones de la población actual. En contraste, en los algoritmos genéticos es un operador secundario que tiene la finalidad de mantener el espacio de búsqueda totalmente conectado.

Es importante mencionar que los teoremas que establecen la propiedad de convergencia al óptimo global de los algoritmos evolutivos se basan en la propiedad de que todos los posibles fenotipos pueden ser generados por los operadores de variación; esto se logra usualmente por medio del operador de mutación [24].

El operador de **recombinación**, también llamado **operador de cruza**, tiene como finalidad transferir de los individuos a sus descendientes la información obtenida durante el proceso de búsqueda. En general, los algoritmos evolutivos no poseen un mecanismo explícito de memoria. La información obtenida durante el proceso de búsqueda es almacenada es los individuos dentro de la población.

El papel del operador de cruza depende del tipo de algoritmo evolutivo empleado. Así, para los algoritmos genéticos la *cruza* es el principal operador de variación, en las estrategias evolutivas es un operador secundario y en la programación evolutiva no se emplea tal

operador.

El principio detrás de la recombinación es simple: cruzar a dos o más individuos con características deseadas para producir uno o más individuos que combinen tales cualidades.

Mecanismo de Supervivencia

En muchos de los casos, las poblaciones de individuos se mantienen a un tamaño constante; esto pretende simular un ambiente con espacio finito donde los individuos y sus descendientes compiten por un lugar en la población. Este mecanismo varía de acuerdo al tipo de algoritmo evolutivo empleado; por ejemplo, en el caso de los algoritmos genéticos, los descendientes reemplazan en su totalidad a los *padres*. En contraste en las estrategias evolutivas los individuos y sus descendientes compiten por un lugar en la población. Este mecanismo se emplea de manera estocástica o determinista.

2.5. Principales Paradigmas de la Computación Evolutiva

La idea de solucionar problemas basándose en los principios de la evolución natural no es nueva; por ejemplo, en 1957 Box [4] propuso un método llamado *Evolutionary Operation* para resolver problemas de optimización industrial. Hoy en día se consideran como los principales paradigmas de la computación evolutiva a tres tipos de técnicas: *algoritmos genéticos, estrategias evolutivas y programación evolutiva.*

2.5.1. Programación Evolutiva

La programación evolutiva fue desarrollada originalmente por Lawrence J. Fogel para simular a la evolución como un proceso de aprendizaje con la finalidad de generar inteligencia artificial [29, 30].

En la programación evolutiva la inteligencia se define como la capacidad de un sistema a adaptarse a su ambiente con el fin de cumplir objetivos específicos.

En las primeras versiones de la programación evolutiva (PE), el ambiente es descrito por una secuencia de símbolos tomados de un alfabeto finito; el algoritmo evolutivo opera sobre la secuencia de símbolos con la finalidad de predecir el siguiente símbolo de la secuencia. Se emplean máquinas de estados finitos para representar a los individuos dentro de la población. No se ocupa un operador de recombinación; las variaciones únicamente son producto de la mutación. La aptitud es una medida basada en la exactitud de la predicción

y la complejidad de los individuos, es decir, las máquinas de estados producidas durante el proceso evolutivo.

Los principales tipos de mutación son: cambiar un símbolo de salida, cambiar un estado de transición, añadir un estado, eliminar un estado, cambiar el estado inicial. La elección del tipo de mutación es efectuada al azar y cada individuo de la población genera un descendiente (producto de la mutación). La población se mantiene de un tamaño fijo; así, se elige a la nueva población seleccionando a las mejores soluciones entre individuos y descendientes.

Fogel et al. [29] describen un experimento donde evolucionan estimadores con la finalidad de determinar si el siguiente número entero en una secuencia es primo o no; la secuencia de símbolos empleada representa a la sucesión de los números naturales $(1, 2, 3, \cdots)$. En los resultados, se muestra que tal sistema alcanza una porcentaje de acierto del 80 %.

Por razones históricas la PE ha sido asociada con tareas de predicción, donde se emplean máquinas de estados finitos como forma de representación de los individuos. Sin embargo, existen otras variantes de la PE (p. ej. en optimización de parámetros con valores reales [5]).

Hoy en día, la PE considera que la representación no debe ser única y por consecuencia la mutación tampoco. Por el contrario, son dependientes del problema a resolver. Así, la PE presenta un esquema muy general para la solución de problemas. En el algoritmo 2 se muestra una descripción general de la programación evolutiva.

```
1 Inicializar(P_g);
2 Evaluar(P_g) ;
3 while Criterio de Terminación NO satisfecho do
4 |   P'_g ← Mutar(P_g) ;
5 |   Evaluar(P'_g) ;
6 |   P_{g+1} ← Seleccionar(P'_g ∪ P_g) ;
7 |   G = G + 1 ;
```

Algoritmo 2: Esquema general de la programación evolutiva

Una de las características principales de la programación evolutiva es que no emplea un operador de recombinación. En el contexto de máquinas de estados finitos, la idea de recombinación no parece obvia. En la PE, un punto en el espacio de búsqueda no se considera como un individuo de una población, sino como una abstracción de una especie. Por tal motivo no se emplea un operador de recombinación.

En la PE no existe un proceso de selección de *padres*, puesto que todos los individuos de la población generan un descendiente. El mecanismo de supervivencia es simple: dada una población de μ individuos se generan μ descendientes; entre las $\mu + \mu$ soluciones se efectúan una serie de torneos con la finalidad de seleccionar μ individuos que forman la siguiente población.

2.5.2. Algoritmos Genéticos

Los algoritmos genéticos fueron propuestos inicialmente por John H. Holland como métodos con comportamiento adaptativo que conservan los mecanismos más importantes de los sistemas naturales [41]. Hoy en día se emplean ampliamente como métodos de búsqueda y optimización.

En los algoritmos genéticos (AG), cada fenotipo forma parte de una población de individuos similares. La población constantemente fluctúa debido a la reproducción y muerte de los miembros que la componen. La influencia de un individuo en el desarrollo futuro de la población depende directamente de su aptitud. De hecho, la aptitud es una medida de la capacidad reproductiva de un individuo. Así, la reproducción se efectúa en proporción a una medida de desempeño; ésta es la forma en que la historia de las soluciones previamente generadas influyen en el futuro de la población. La población sirve como un repositorio de información genotípica con características deseadas.

En la forma canónica de los algoritmos genéticos, los individuos son representados como una secuencia de símbolos de un alfabeto dado, Holland sugiere emplear la codificación binaria [41]. La codificación representa el genotipo del individuo.

La mutación juega un papel secundario, dado que las variaciones completamente aleatorias en los genotipos pueden destruir la información contenida en la población. Así, el operador principal es la recombinación (**cruza**). La cruza redistribuye los segmentos de información genotípica de los individuos a sus descendientes. Esta cruza opera de manera constructiva, permitiendo que los segmentos de información *útil* se propaguen a lo largo de las generaciones.

Así, las principales características de los algoritmos genéticos son:

- Una población de individuos que retiene la información obtenida durante el proceso de búsqueda y que sirve como fuente a nuevas variaciones.
- La capacidad reproductiva de las soluciones es proporcional a un valor de desempeño (aptitud).

- Los operadores de recombinación promueven la propagación de información *útil* de los individuos a sus descendientes.

Actualmente, los algoritmos genéticos proporcionan un marco de trabajo para el desarrollo de métodos de solución a problemas específicos. Existen en la literatura una amplia variedad de tipos de representación, de operadores de cruza y/o mutación, de procedimientos de asignación de aptitudes, de métodos de reproducción, nuevos tipos de operadores, etc.

Los tipos de elementos que conforman a un algoritmo genético son dependientes del problema. En el algoritmo 3 se muestra en pseudocódigo un algoritmo genético en su forma canónica.

```
1  Inicializar(P_g) ;
2  Evaluar(P_g) ;
3  while Criterio de Terminación NO satisfecho do
4      AsignarProbabilidadReproduccion(P_g) ;
5      P'_g ← Reproducir(P_g) ;
6      P''_g ← Cruzar(P'_g) ;
7      P'''_g ← Mutar(P''_g) ;
8      Evaluar(P'''_g) ;
9      P_{g+1} ← P'''_g ;
10     G ← G + 1 ;
```

Algoritmo 3: Esquema de un algoritmo genético canónico

Cabe mencionar que los algoritmos genéticos son el tipo de algoritmo evolutivo más ampliamente utilizado. Las aplicaciones de los AG son muy diversas; entre ellas están [32]: optimización numérica y combinatoria, aprendizaje de máquina, optimización de consultas en bases de datos, reconocimiento de patrones, generación de gramáticas, planeación de movimientos de robot, etc.

2.5.3. Estrategias Evolutivas

Las Estrategias Evolutivas (EE) son otro enfoque en la simulación de la evolución. Éstas se desarrollaron en los años 60's por Hans-Paul Schwefel e Ingo Rechenberg, y fueron diseñadas en el ámbito de la optimización de funciones continuas.

Los componentes de una solución se consideran *características de comportamiento* de un individuo y no genes en un cromosoma. Existe una fuente genética para estas características

fenotípicas, pero la naturaleza de tal fuente no es necesario describirla. Se asume que, dondequiera que la transformación genética ocurra, los cambios resultantes en las características de comportamiento estarán determinados por una distribución normal de probabilidades con media cero y alguna desviación estándar.

Las alteraciones genéticas pueden afectar a más de una característica fenotípica debido a la pleitropía y a la poligenia; de tal manera, es necesario variar simultáneamente todos los componentes de un individuo durante el proceso de generación de los descendientes.

En el algoritmo 4 se muestra el algoritmo básico de una estrategia evolutiva de dos miembros para un problema de minimización de una función $n-$dimensional $f : \mathbb{R}^n \to \mathbb{R}$ [84]. $N(0, \sigma)$ representa la generación de un número aleatorio conforme a una distribución normal de probabilidades con media cero y desviación estándar σ.

```
1 t = 0 ;
2 Generar de manera aleatoria ⟨x_1^t, ··· , x_n^t⟩ ∈ ℝ^n ;
3 while Criterio de Terminación NO satisfecho do
4 |   y_i^t = x_i^t + N(0, σ) para todo i ∈ {1, ··· , n} ;
5 |   if f(ȳ^t) ≤ f(x̄^t) then
6 |   |   x̄^{t+1} = ȳ^t
7 |   else
8 |   |   x̄^{t+1} = x̄^t
9 |   t = t + 1 ;
```

Algoritmo 4: Estrategia evolutiva de dos miembros $EE - (1 + 1)$

La mutación se genera perturbando la solución actual usando una distribución normal con media cero y desviación estándar σ (*tamaño de paso*). Esto genera una distribución simétrica en todos los n componentes de $\vec{x}^t$. Estudios teóricos proponen un ajuste en línea del *tamaño de paso* usando la **regla del éxito** 1/5. Esta regla establece que la razón de mutaciones exitosas (el descendiente es más apto que el padre) con respecto a todas las mutaciones realizadas debe ser $\frac{1}{5}$. Así, si la razón es mayor a $\frac{1}{5}$ el tamaño de paso debe ser incrementado y si la razón es menor a $\frac{1}{5}$ el valor de σ debe ser decrementado; de otra manera, no hay cambio.

Hoy en día, las estrategias evolutivas casi siempre usan *auto-adaptación*; así, el vector $\vec{x} = \langle x_1^t, \cdots, x_n^t \rangle$ es sólo un componente de la solución. En general un individuo es representado de la siguiente forma.

$$\langle x_1, \cdots, x_n, \sigma_1, \cdots, \sigma_n, \alpha_1, \cdots, \alpha_{n(n-1)/2} \rangle$$

Donde σ_i para $i = \{1, \cdots, n\}$ son los tamaños de paso para cada uno de los n componentes de $\vec{x}$, y α_i para $i = \{1, \cdots, \frac{n(n-1)}{2}\}$ representan las interacciones entre los diferentes tamaños de pasos.

Existen estrategias evolutivas multi-miembro [84]. En tales casos, a partir de μ individuos se generan λ descendientes. Existen básicamente dos esquemas de supervivencia: en la estrategia $(\mu + \lambda)$ se selecciona a los μ mejores individuos de la unión de los μ padres y sus λ hijos; en la estrategia (μ, λ) se selecciona a los μ mejores individuos de entre los λ hijos producidos. En ambos casos la selección es determinista.

En las EE el operador de variación principal es la mutación. La recombinación (cruza) es un operador secundario y sólo se utiliza en algunos casos. Cuando el operador de cruza es empleado, la selección de los *padres* no se basa en el valor de aptitud sino que, se seleccionan los *padres* de manera aleatoria de la población.

Las estrategias evolutivas han sido empleadas para resolver problemas tales como [83]: ruteo de redes, problemas de bioquímica, problemas de óptica, problemas de diseño en ingeniería, etc.

2.6. Nuevas Alternativas

Hoy en día, existe un creciente interés por simular procesos naturales, sin limitarse a los basados en la evolución de las especies. Dichas simulaciones pretenden ofrecerse como métodos alternativos de solución a una amplia gama de problemas. Algunos de estos métodos se listan a continuación.

Programación Genética Es una extensión al modelo del algoritmo genético empleado para el aprendizaje de máquina. La programación genética pretende evolucionar poblaciones de programas de computadora que mejoren automáticamente a través de la experiencia (aprendizaje inductivo).

Optimización con Colonia de Hormigas Estos algoritmos son sistemas multi agentes en los cuales, el comportamiento de cada agente, llamado *hormiga*, está inspirado en el proceder natural de las hormigas biológicas. Los algoritmos de colonia de hormigas son uno de los más exitosos ejemplos de sistemas de comportamiento emergente (*swarm intelligence systems*) [23].

Estos algoritmos han sido aplicados principalmente en problemas de optimización combinatoria; por ejemplo, en el problema del agente viajero, en ruteo de redes de telecomunicaciones, en el problema de la asignación cuadrática, en la coloración de grafos, etc. [23].

Sistema Inmune Artificial Los sistemas inmunes artificiales simulan uno o más componentes funcionales del sistema inmune natural. El sistema inmune natural es un sistema complejo y adaptativo. Visto desde una perspectiva de procesamiento de información, éste es un sistema inteligente altamente paralelo, que emplea aprendizaje, memoria y asociación para resolver problemas de clasificación y reconocimiento. Por tal motivo, ha sido objeto de estudio e inspiración para diversos modelos computacionales [17].

Algunas de sus aplicaciones son: seguridad de computadoras, detección de virus, detección de anomalías, reconocimiento de patrones, optimización, etc. [17].

Algoritmos Meméticos Los algoritmos meméticos son estrategias *cooperativas-competitivas* de agentes que optimizan. La idea de los algoritmos meméticos es emplear la **sinergia** de diferentes métodos de búsqueda. El término *algoritmos meméticos* identifica una amplia clase de técnicas de búsqueda donde el conocimiento del problema es incorporado al método de búsqueda en la forma de heurísticas, operadores especiales, buscadores locales, etc. [68].

Estas técnicas han sido aplicadas exitosamente en la solución de problemas combinatorios, particularmente en la aproximación de soluciones a problemas NP [68].

Optimización con Cúmulos de Partículas El algoritmo de optimización con cúmulos de partículas modela la exploración del espacio de búsqueda por medio de una población de individuos donde el éxito individual influye en el comportamiento de sus semejantes. Esta técnica se originó como una simulación del comportamiento social de los individuos [48].

Ésta es una de las heurísticas bio-inspiradas más usadas actualmente donde ha tenido aplicaciones muy diversas, sobre todo en problemas de optimización cuyas variables de decisión son números reales.

Evolución Diferencial Es un algoritmo eficiente empleado en optimización global de funciones en espacios totalmente ordenados [75]. Este trabajo de tesis está basado en esta heurística, cuyo funcionamiento se detallará en capítulos posteriores.

2.7. Resumen del Capítulo

La Computación Evolutiva es un área de las ciencias de la computación inspirada en los principales procesos de la evolución natural: *reproducción*, *mutación*, *competencia* y *selección*. La evolución natural se observa como un proceso de solución a problemas, los cuales están caracterizados por el caos, el azar, la temporalidad y la no linealidad. Éstas son a su vez las características de los problemas que han demostrado ser especialmente intratables por métodos clásicos. Así, los algoritmos evolutivos ofrecen una opción de solución a problemas donde los métodos clásicos no ofrecen un resultado satisfactorio dado un conjunto limitado de recursos de cómputo.

Los principales paradigmas de la computación evolutiva son: los algoritmos genéticos empleados ampliamente en búsqueda y optimización, las estrategias evolutivas diseñadas para la optimización numérica y la programación evolutiva desarrollada para el aprendizaje de máquina.

Hoy en día se han desarrollado otras heurísticas bio-inspiradas. Entre las más exitosas se encuentran: la optimización con cúmulos de partículas, el sistema inmune artificial, la optimización con colonia de hormigas y la evolución diferencial.

Capítulo 3

Nociones de Optimización

En este capítulo se presentan las nociones básicas de optimización, se establece el planteamiento del problema que atañe a esta tesis y se listan las características de los problemas de optimización restringida. Además, se muestran las principales diferencias entre la optimización con y sin restricciones.

3.1. Introducción

El diccionario de la Real Academia de la Lengua Española define el término **optimizar** como:

Buscar la mejor manera de realizar una actividad

En el área de optimización en ingeniería, ésta se define como *el acto de obtener los mejores resultados bajo ciertas circunstancias* [76].

En el contexto de este trabajo, la **optimización** se define como *el proceso de buscar la mejor solución posible a un problema, bajo ciertas circunstancias.*

Un problema de optimización es un planteamiento a una situación para la cual existen diferentes soluciones posibles y hay una clara noción de calidad de las soluciones. Los problemas de optimización existen cuando diferentes soluciones candidatas pueden ser claramente comparadas y contrastadas.

Como se dijo antes, no existe un solo método capaz de resolver todos los problemas de optimización de manera eficiente [76]. Por tal motivo, una gran cantidad de métodos de optimización han sido desarrollados para resolver diferentes tipos de problemas. Estos métodos son comúnmente nombrados *técnicas de programación matemática* y forman parte

de una área de las matemáticas llamada *investigación de operaciones*.

Entre las aplicaciones típicas de la optimización en ingeniería se encuentran: diseño de estructuras, cálculo de trayectorias de vehículos espaciales, diseño de turbinas, diseño de redes eléctricas, diseño de planes de producción, diseño de procesos químicos, etc. [76].

3.2. Planteamiento del Problema

El problema que atañe a esta tesis es denominado *problema general de programación no lineal* (PGPNL); y se describe a continuación.

$$\text{Encontrar } \vec{x} \text{ tal que optimice a } f(\vec{x})$$

Sujeto a las siguientes restricciones:

$$\begin{aligned} g_i(\vec{x}) &\leq 0, \quad i = 1, \cdots, m \\ h_j(\vec{x}) &= 0, \quad j = 1, \cdots, p \end{aligned}$$

El vector de diseño $\vec{x} = (x_1, \cdots, x_n)$ es una $n-$tupla, donde $x_i (i = 1, \cdots, n)$ son valores escalares llamados **variables de decisión o variables de diseño**. En este trabajo de tesis se consideran únicamente instancias del PGPNL donde $\vec{x} \in \mathbb{R}^n$.

Se llama **espacio de diseño o espacio de búsqueda** $\mathcal{S}$ al espacio cartesiano $n-$dimensional donde cada uno de los ejes coordenados representa una variable de decisión $x_i (i = 1, \cdots, n)$; cada punto en él, es llamado **punto de diseño** y representa una solución posible (aunque quizás infactible) al problema.

$f : \mathbb{R}^n \to \mathbb{R}$ es una función escalar y es denominada **función objetivo**. En este trabajo no se requiere que f tenga propiedades especiales, únicamente que $f(\vec{x})$ exista para cualquier punto en el espacio de diseño.

$g_i(\vec{x})$ para $i = \{1, \cdots, m\}$ y $h_j(\vec{x})$ para $j = \{1, \cdots, p\}$ son llamadas restricciones de desigualdad y de igualdad respectivamente; $g_i : \mathbb{R}^n \to \mathbb{R}$ y $h_j : \mathbb{R}^n \to \mathbb{R}$ son funciones escalares y no tienen propiedades especiales al igual que f. En general, f, g_i y h_j son no lineales.

La región factible $\mathcal{F} \subseteq \mathcal{S}$ representa las *soluciones posibles* al problema y se define como:

$$\mathcal{F} = \{\vec{x} \in \mathbb{R}^n | \quad g_i(\vec{x}) \leq 0,\ i = \{1, \cdots, m\} \text{ y } h_j(\vec{x}) = 0,\ j = \{1, \cdots, p\}\}$$

Se dice que $g_i(\vec{x})$ para $\vec{x} \in \mathcal{F}$ es una restricción **activa** cuando $g_i(\vec{x}) = 0$ en el óptimo. En caso contrario, se denomina restricción **inactiva**. Todas las restricciones de igualdad se consideran *activas* en todos los puntos de la región factible.

En optimización en ingeniería se clasifican a las restricciones en dos tipos: las restricciones que representan limitaciones de desempeño o de comportamiento son denominadas **restricciones funcionales o suaves** y las restricciones que representan limitaciones físicas de diseño son denominadas **restricciones laterales o duras**.

Considerando únicamente problemas de **minimización** (sin pérdida de generalidad) se tienen las siguientes definiciones.

Definición 1 *Una función $f(\vec{x})$ definida en un conjunto $\mathcal{S}$ posee su óptimo global en el punto $\vec{x}^* \in \mathcal{S}$ si y sólo si: $f(\vec{x}^*) \leq f(\vec{x})$ para todo $\vec{x} \in \mathcal{S}$*

Definición 2 *Una función $f(\vec{x})$ definida en un conjunto $\mathcal{S}$ posee un óptimo local (mínimo relativo) en el punto $\vec{x}^l \in \mathcal{S}$ si y sólo si: $f(\vec{x}^l) \leq f(\vec{x})$ para todo $\vec{x}$ a una distancia ϵ de $\vec{x}^l$. Esto es, existe un $\epsilon > 0$ tal que para todo $\vec{x}$ que satisface $|\vec{x} - \vec{x}^l| < \epsilon$, $f(\vec{x}^l) \leq f(\vec{x})$.*

Así, $\vec{x}^*$ y $f(\vec{x}^*)$ representan al punto y valor óptimo respectivamente; a este par, se le denomina **solución óptima**.

3.3. Características de los Problemas Restringidos

En la presencia de restricciones, un problema de optimización puede presentar las siguientes características:

Las restricciones pueden no tener efecto en el *punto óptimo*. Esto es, el valor óptimo del problema restringido es el mismo que el del problema sin restricciones tal como se muestra en la figura 3.1.

En tal caso, se podrían emplear métodos de optimización sin restricciones para su solución. Sin embargo, en la mayoría de los problemas es extremadamente difícil identificar tal situación. Así, se establece la suposición de que las restricciones tienen influencia en el

óptimo.

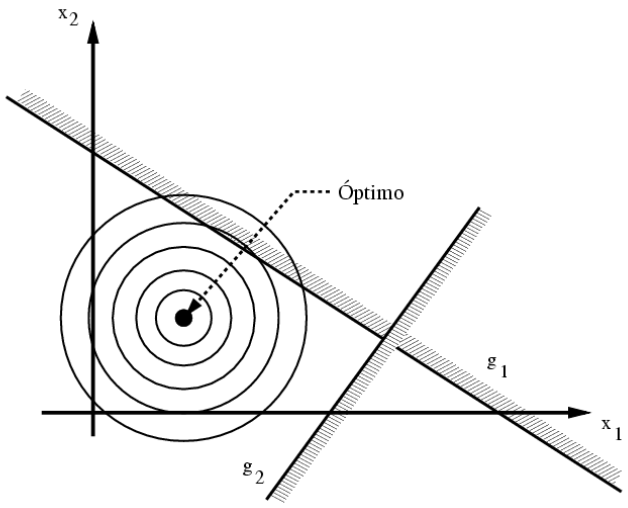


Figura 3.1: El óptimo es el mismo con o sin restricciones

La solución única puede tener lugar en el borde de la zona factible. En este caso, las restricciones influyen en la localización del punto óptimo. Este efecto se observa en la figura 3.2, donde el óptimo global de la función (también llamado óptimo irrestricto) no se localiza dentro de la zona factible.

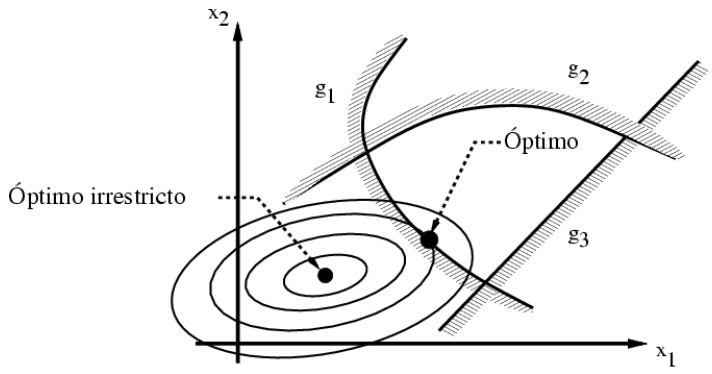


Figura 3.2: El óptimo difiere debido a las restricciones

Las restricciones pueden incrementar el número de óptimos locales, aún cuando la función objetivo posea un único óptimo global. Esto se observa en la figura 3.3.

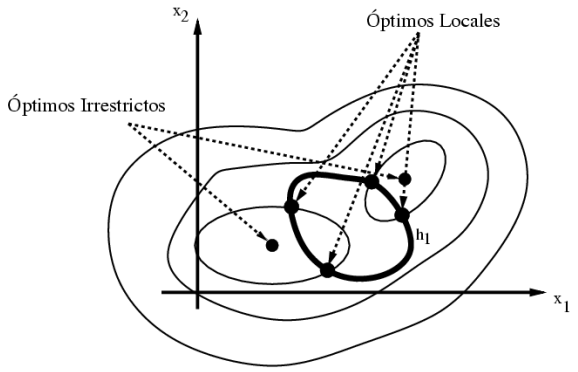


Figura 3.3: Situación en la que se crean óptimos locales a consecuencia de incluir restricciones al problema

Harold Kuhn y Albert Tucker desarrollaron las condiciones que deben satisfacer los óptimos relativos de $f(\vec{x})$. En el caso general, estas condiciones no aseguran óptimos relativos. Sin embargo, para una clase de problemas llamados *problemas de programación convexa*, las condiciones de Kuhn-Tucker son necesarias y suficientes para establecer un óptimo global [76].

En los problemas de programación convexa la función objetivo $f(\vec{x})$ y las funciones de restricción $g(\vec{x})$ son convexas.

Definición 3 *Una función $f(\vec{x})$ se denomina función convexa si para cualesquiera pares de puntos $\vec{x}_1, \vec{x}_2$ y para todo $0 \leq \lambda \leq 1$.*

$$f(\lambda\vec{x}_2 + (1-\lambda)\vec{x}_1) \leq \lambda f(\vec{x}_2) + (1-\lambda)f(\vec{x}_1)$$

Esto es, el segmento que une a los dos puntos yace completamente arriba de la gráfica de $f(\vec{x})$.

Las condiciones de Kuhn-Tucker requieren que f, g_i, h_j sean diferenciables. Además, en el caso de la condición de suficiencia las restricciones de igualdad (h_j) deben ser lineales.

Existen condiciones *de necesidad y de suficiencia* que no requieren la convexidad de f, g_i, h_j y tampoco que $h_j (j = 1, \cdots, p)$ sean lineales. Estas condiciones son llamadas de

segundo orden. Sin embargo, éstas requieren que f, g_i, h_j sean doblemente diferenciables [60].

Por tal motivo, aunque se han desarrollado técnicas de programación matemática para resolver problemas con propiedades particulares (p. ej. programación lineal, programación cuadrática, etc.) el problema general de la programación no lineal (con o sin restricciones) sigue siendo abierto y, dada su complejidad, el uso de técnicas heurísticas (como la adoptada en esta tesis) está plenamente justificado [76].

3.4. Resumen del Capítulo

La optimización se define como el proceso de buscar la mejor solución posible a un problema bajo ciertas circunstancias. Desafortunadamente no existe un solo método capaz de resolver todos los problemas de optimización de manera eficiente. Por tal motivo, hoy en día la optimización es una área activa de investigación.

En la presencia de restricciones, un problema de optimización puede presentar las siguientes características: las restricciones no tienen efecto en el punto óptimo, las restricciones influyen en la localización del punto óptimo o las restricciones pueden incrementar el número de óptimos locales.

No existen, en el caso general condiciones de necesidad o suficiencia que determinan la optimalidad de una solución. El problema general de la programación no lineal sigue siendo abierto y, dada su complejidad, el uso de técnicas heurísticas (como la adoptada en esta tesis) está plenamente justificado.

Capítulo 4

Evolución Diferencial

En este capítulo se presenta una descripción completa del algoritmo de la Evolución Diferencial mostrando las principales características de cada uno de sus componentes.

Además, se efectúa un estudio experimental con el fin de determinar las características y los comportamientos reales de los principales modelos de la Evolución Diferencial propuestos en la literatura especializada.

4.1. Introducción

La Evolución Diferencial (ED) es una técnica estocástica de búsqueda directa la cual ha mostrado ser un método *robusto* de búsqueda en una amplia gama de problemas [75].

La ED fue propuesta como una nueva heurística para la minimización de funciones no lineales y no diferenciables en espacios totalmente ordenados, la cual fue presentada por Rainer Storn y Kenneth Price en 1995 [85].

El primer paso para resolver un problema de optimización es determinar si existe un algoritmo diseñado específicamente para el problema en cuestión. Por ejemplo, si la función objetivo es un polinomio lineal existen métodos eficientes (p. ej. método simplex [76]) que pueden proveer soluciones globales a instancias del problema que contienen cientos o inclusive miles de variables de decisión. Desafortunadamente, los algoritmos tradicionales deterministas son insuficientes cuando la función objetivo posee características como: no linealidad, alta dimensionalidad, existencia de múltiples óptimos locales (multimodalidad), no diferenciabilidad o ruido.

La ED es un método alternativo en la solución a problemas con estas características, empleando un novedoso esquema simple de mutación auto-adaptada.

La ED surgió de los intentos de Kenneth Price por resolver el problema del *ajuste polinomial de Chebychev*. Con el fin de mejorar las soluciones existentes, se emplearon vectores diferencia como mecanismo de perturbación y de esta idea surgió la ED.

La comunidad de la ED ha estado creciendo desde su aparición y cada vez hay más investigadores trabajando en ella. Esto hace de la ED una muy área activa de investigación hoy en día.

Dentro de las características principales de la ED se encuentran [86, 75]:

- La capacidad de manejar funciones objetivo no lineales, no diferenciables y multimodales.
- El algoritmo es fácilmente paralelizable y resulta útil cuando la evaluación de la función objetivo es computacionalmente costosa.
- No se requiere predefinir distribuciones de probabilidad como en el caso de las estrategias evolutivas [84].
- Emplea una codificación real y la precisión está determinada por el formato de punto flotante empleado.
- Suele converger a un valor óptimo (posiblemente local) de manera consistente a lo largo de una secuencia de ejecuciones independientes.
- En su forma original, el algoritmo emplea únicamente tres parámetros de control además de un criterio de terminación.

Asimismo, la ED posee una amplia gama de aplicaciones, de entre las cuales se encuentran [54, 2]: diseño de filtros digitales, entrenamiento de redes neuronales, diseño óptimo de tubos para intercambio de calor, optimización de procesos químicos no lineales, optimización de tuberías de agua, diseño de redes de transmisión de gas, calendarización óptima de satélites, etc.

4.2. Descripción del Algoritmo

La Evolución Diferencial es un algoritmo evolutivo típico que cae en la descripción del esquema 1 mostrado en la página 10 de esta tesis. Como en la mayoría de los algoritmos

evolutivos, la Evolución Diferencial emplea una población de individuos que representan soluciones candidatas al problema en cuestión.

Las heurísticas que incorporan información del dominio inevitablemente obtienen un rendimiento superior a sus contrapartes que no lo hacen [75]. Por tal motivo, los algoritmos evolutivos canónicos presentan un menor rendimiento en comparación a versiones híbridas que sí incorporan información del problema (p. ej. empleando buscadores locales [67] o algoritmos culturales [77]).

La ED está restringida a dominios donde el espacio de búsqueda está completamente ordenado y en especial subespacios de $\mathbb{R}^n$. En la ED, un individuo se representa como una n-tupla llamada vector objetivo $\vec{x} = (x_1, \cdots, x_n)$, donde $x_i \in \mathbb{R}$ $(i = 1, \cdots, n)$ son valores escalares que representan las variables de diseño del problema. Una forma de incorporar información del dominio es restringir a esta clase de problemas a la ED. Esto sin comprometer su estatus de optimizador numérico de propósito general.

Al emplear una representación real de los individuos, en lugar de símbolos de un alfabeto dado como en los algoritmos genéticos, se evitan los problemas que surgen entre el mapeo no consecutivo del genotipo y el fenotipo [21]. Por ejemplo, empleando una codificación binaria tradicional dos genotipos consecutivos no necesariamente corresponden a fenotipos contiguos; así, no se obtiene beneficio alguno de que $\mathbb{R}^n$ sea un espacio totalmente ordenado. Por otro lado, emplear una codificación real permite generar perturbaciones acotadas en las variables de diseño, esto a consecuencia del orden determinado por $\mathbb{R}^n$.

Aún empleando una codificación de códigos de gray (donde el mapeo es consecutivo) es necesario usar secuencias de símbolos muy largas cuando los rangos de las variables de decisión son amplios o dinámicos. Además, la mayoría de estos símbolos no son significativos en el resultado final.

A continuación se listan algunas de las ventajas de emplear una codificación real:

- Permite emplear dominios más grandes, e inclusive dominios no acotados.
- Permite utilizar grados de precisión más exactos.
- Permite explotar la *granularidad* del paisaje de aptitud.
- Se elimina el proceso de codificación / decodificación.
- El nivel de expresividad es más alto, lo cual coincide con la intuición de que una representación más expresiva nos provee de un aparato más poderoso de adaptación.

- Permite integrar información del dominio de manera más simple (p. ej. para manejo de restricciones).

Por tales motivos, la ED emplea una codificación real donde las limitaciones de rango y precisión son propias del formato de coma flotante empleado y no del algoritmo en sí.

En su versión original, la ED emplea una población de tamaño fijo. Esto es por conveniencia y no existe razón *a priori* del porqué de esta elección [75]. Existe otra propuesta donde el tamaño de la población se codifica como un parámetro de diseño; así, la población aumenta o disminuye dinámicamente [90]. Sin embargo, esta propuesta no muestra mejoras significativas con respecto a la versión original.

Como primer paso del algoritmo, se generan aleatoriamente np individuos que conforman a la población inicial $P_0 = \{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_{np}\}$. En la mayoría de los casos los individuos están uniformemente distribuidos en el espacio de búsqueda dentro de los límites definidos, como se muestra en la ecuación (4.1).

$$x_{i,j} = \underline{x_j} + U(0,1) \cdot (\overline{x_j} - \underline{x_j}) \quad \forall i \in \{1, \ldots, np\},\ \forall j \in \{1, \ldots, n\} \tag{4.1}$$

$x_{i,j}$ representa a la j-ésima variable de decisión del i-ésimo individuo en la población, $U(0,1)$ representa una variable aleatoria uniformemente distribuida en el rango $[0,1]$ y $\overline{x_j}, \underline{x_j}$ representan los límites superiores e inferiores respectivamente de la j-ésima variable de diseño. Los límites son determinados por restricciones físicas del problema; de otra manera, éstos deben cubrir la región donde se espera que esté el valor óptimo.

Si está disponible una solución aproximada al problema, entonces se pueden generar los individuos iniciales perturbando a la solución dada empleando una función de distribución normal. Es importante mencionar que los individuos iniciales deben estar distribuidos aleatoriamente de alguna manera ya que las diferencias de los vectores objetivo conducen el comportamiento de la ED.

Después de la inicialización, la población es sujeta a un proceso iterativo de *mutación, recombinación y selección* durante G_{max} iteraciones (generaciones). Como se muestra en la ecuación (4.2), la ED emplea los operadores de mutación y recombinación para generar un solo descendiente o *vector candidato* $\vec{u}_i$ por cada vector *padre* $\vec{x}_i$ que pertenece a la población.

$$u_{i,\forall j\in\{1,\dots,n\}} = \begin{cases} x_{r_3,j} + F\cdot(x_{r_1,j} - x_{r_2,j}) & \text{si } U(0,1) < CR \text{ o } j = j_{rand} \\ x_{i,j} & \text{en caso contrario} \end{cases} \tag{4.2}$$

Donde $r_1, r_2, r_3 \in \{1, \cdots, np\}$ y $j_{rand} \in \{1, \cdots, n\}$ son seleccionados aleatoriamente, además $r_1 \neq r_2 \neq r_3 \neq i$. CR y F son variables de control del algoritmo y son proporcionadas por el usuario. CR representa una probabilidad y su rango es $[0, 1]$, F es un factor de escalamiento cuyo intervalo es típicamente $[0, 1)$. Se observa que cuando $U(0,1) < CR$ o $j = j_{rand}$ el valor de la j-ésima variable de decisión es una combinación lineal de los vectores aleatoriamente seleccionados $(\vec{x}_{r_1}, \vec{x}_{r_2}, \vec{x}_{r_3})$; de otra manera, este valor es *heredado* directamente del *padre*. La condición $j = j_{rand}$ se incluye con el fin de asegurar que $\vec{u}_i$ difiera al menos en un parámetro de diseño de $\vec{x}_i$.

Después de que cada $\vec{u}_i$ es evaluado en la función objetivo, los valores de $f(\vec{u}_i)$ y $f(\vec{x}_i)$ son comparados. Se elige como $\vec{x}_{i,G+1}$ a quien posea un mejor valor de aptitud dependiendo de si el problema es de minimización o maximización.

Los procesos de recombinación y selección para una solución $\vec{x}_i$ durante la generación G se ilustran en la figura 4.1, donde la *cruza discreta* corresponde a la elección estocástica mostrada en la ecuación (4.2).

Una vez determinados los individuos que conforman la siguiente población, el ciclo se repite hasta que el problema sea resuelto o todos los vectores objeto converjan a un punto o no se logre ninguna mejora después de varias iteraciones o después de ejecutarse un número fijo de generaciones. El algoritmo 5 muestra a la Evolución Diferencial en su forma original.

4.3. Componentes de la Evolución Diferencial

4.3.1. Esquema de Mutación

En el contexto de la optimización de parámetros reales, la mutación se refiere al proceso de agregar un incremento generado aleatoriamente a una o más variables de decisión de un vector solución dado. El objetivo de un esquema eficiente de mutación es generar incrementos o movimientos que desplacen a los vectores solución en la dirección y magnitud correcta. Lograr este objetivo depende directamente de las características de la función de distribución empleada en la generación de las perturbaciones. El problema es que no existe una sola función de distribución que posea la versatilidad requerida para explorar eficiente-

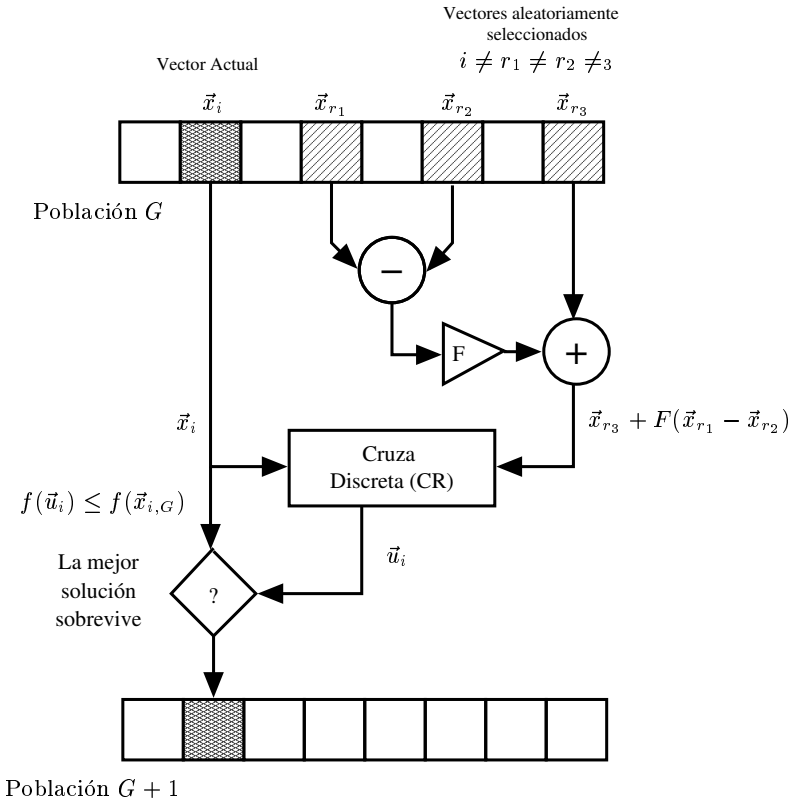


Figura 4.1: Diagrama del algoritmo de Evolución Diferencial

mente todas las funciones objetivo.

Las funciones de distribución empleadas en la generación de perturbaciones no deben mostrar un sesgo natural, es decir, deben exhibir una estadística media de cero. Generar mutaciones con una distribución sin sesgo elimina la posibilidad de que los vectores objeto adquieran una desviación producto de la función de distribución y no como consecuencia de la forma del paisaje de aptitud del problema [75].

Con el fin de ilustrar algunos principios de diseño de la ED, se presentan a continuación una serie de elementos que determinan la eficiencia de las funciones de distribución empleadas en la generación de mutaciones.

Las estrategias evolutivas (EE) son un tipo de algoritmo evolutivo diseñado para la optimización de funciones continuas, en las cuales la mutación es el principal operador de

```
1  G ← 0;
2  Inicializar(P_g ← {x⃗_1, ··· , x⃗_np})
3  while Criterio de Terminación NO satisfecho do
4      for i ← {1, ··· , np} do
5          r_1, r_2, r_3 ∈ {1, ··· , np} aleatoriamente seleccionados,
6                                  donde r_1 ≠ r_2 ≠ r_3 ≠ i ;
7          j_rand ∈ {1, ··· , n} aleatoriamente seleccionado ;
8          for j ← {1, ··· , n} do
9              if U_j[0, 1] < CR or j = j_rand then
10                 u_{i,j} ← x_{r_3,j,G} + F(x_{r_1,j,G} − x_{r_2,j,G}) ;
11             else
12                 u_{i,j} ← x_{i,j,G} ;
13         if f(u⃗_i) ≤ f(x⃗_{i,G}) then
14             x⃗_{i,G+1} ← u⃗_i ;
15     G ← G + 1;
```

Algoritmo 5: Algoritmo de Evolución Diferencial (ED/rand/1/bin)

variación y sólo en algunos casos se emplea un operador de recombinación [25]. En su forma más simple ($EE-(1+1)$), la EE emplea una función de distribución normal para perturbar a una única solución. La nueva solución sustituye a la anterior si presenta un mejor valor de aptitud. El aspecto más importante de la EE-$(1+1)$ es que utiliza un único valor de desviación estándar para perturbar todas las variables de diseño, lo cual presenta algunos problemas.

Supongamos que la función objetivo corresponde a un elipsoide cuyas curvas de nivel se muestran en la figura 4.2.

Para un vector dado $\vec{x}_i$ existen rangos de mutación bien definidos que generan mejores soluciones; esto se muestra en la figura 4.3. Cualquier movimiento de $\vec{x}_i$ a un punto interior de la curva de nivel conducirá a un vector con menor costo. Como se observa en la figura, los intervalos de mejora son diferentes para cada coordenada. Se producen mejores vectores ejecutando movimientos amplios a lo largo del eje mayor pero, como se muestra en la figura, un esquema de mutación simple produce vectores fuera del rango de mejora a lo largo del eje menor del elipsoide. Se observa también que no existe un tamaño de paso que busque

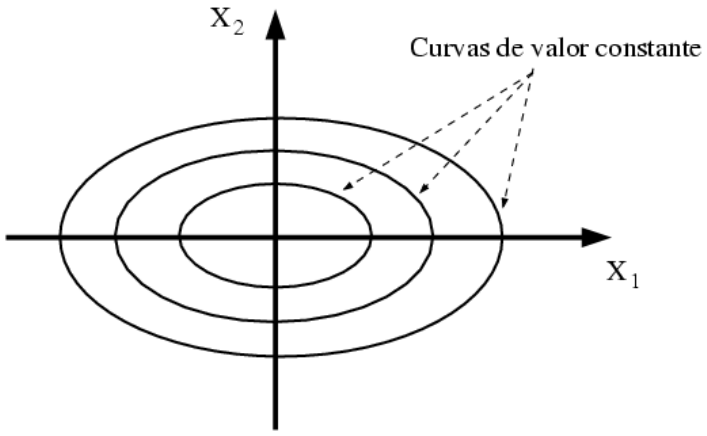


Figura 4.2: Curvas de nivel de elipsoide hipotético

eficientemente a lo largo de todos los ejes coordenados. En el caso de la mutación simple es necesario disminuir el tamaño de paso (desviación estándar) con el fin de encontrar un rango efectivo de mejora en todas las variables de decisión. Esto tiene como consecuencia una disminución en el rendimiento.

En la figura 4.3 se observa la necesidad de emplear una función de distribución que permita diferentes rangos de movimiento en cada coordenada. Las EE emplean la auto-adaptación para manejar dinámica e independientemente los rangos de mutación de cada una de las variables de diseño por cada individuo en la población.

No sólo los tamaños de paso de la mutación necesitan ser ajustados, sino también sus orientaciones. En la figura 4.4 se muestra el caso donde el sistema coordenado del elipsoide es rotado un cierto ángulo. En este caso, una mutación estándar $x_{i,j} + N_j(0, \sigma_{i,j})$ no puede emplear los intervalos máximos de mejora sobre los ejes $x_1^{'}, x_2^{'}$ aun cuando se emplea un tamaño de paso $\sigma_{i,j}$ independiente para cada solución y para cada variable de diseño [75].

La rotación decrementa significativamente los intervalos de mejora a lo largo de los ejes coordenados, retardando el proceso de optimización.

La rotación no sólo demora el proceso de optimización, sino que también elimina la posibilidad de obtener mejoras con movimientos independientes a lo largo de los ejes coordenados. Por tal motivo, es necesaria la generación de perturbaciones simultáneas a lo largo de todos los ejes coordenados. Esta dependencia es llamada **epístasis** y está presente en la mayoría de los problemas del mundo real [75].

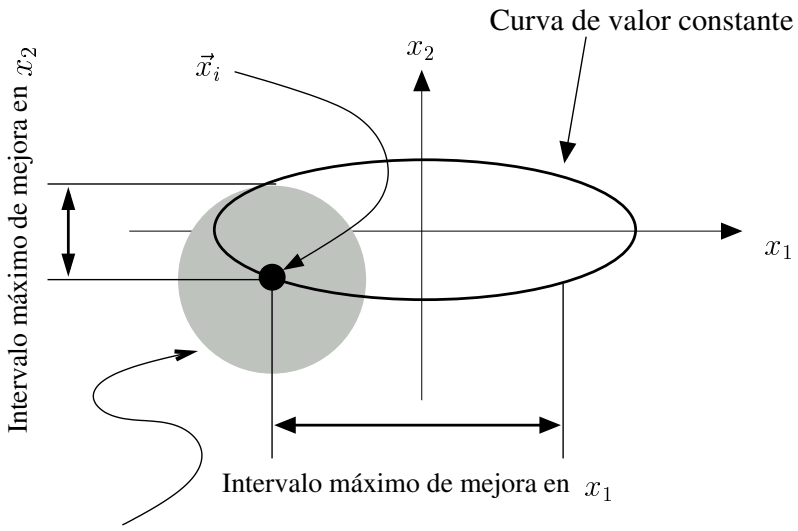


Figura 4.3: Ejemplo de intervalos de mejora usando una función de distribución normal simple

Los algoritmos evolutivos deben ser capaces de perturbar todos los parámetros con mutaciones correlacionadas. Las EE logran este objetivo incorporando ángulos de rotación que determinan la orientación de la mutación. Así, un individuo está compuesto por n variables de diseño, n tamaños de paso y $\frac{n(n-1)}{2}$ coeficientes de mutaciones correlacionadas.

Las EE generan vectores de mutación adaptando dinámicamente un función de distribución multivariable predefinida, mientras que la ED simplemente muestrea aleatoriamente individuos dentro de la población [75]. En su forma más simple la ED genera un vector de mutación sumando una diferencia ponderada de un par de vectores aleatoriamente seleccionados de la población como se muestra en la ecuación (4.3).

$$\vec{u}_i = \vec{x}_i + F \cdot (\vec{x}_{r_1} - \vec{x}_{r_2}) \tag{4.3}$$

La ecuación (4.3) muestra similitud con la *cruza aritmética*, la cual se muestra en la ecuación (4.4).

$$\vec{u}_i = \vec{x}_i + \lambda \cdot (\vec{x}_{r_3} - \vec{x}_i) \tag{4.4}$$

A pesar de la similitud, el vector de mutación de la ED juega un papel muy diferente al de la cruza aritmética. La ecuación (4.4) limita la búsqueda a lo largo del eje que conecta a

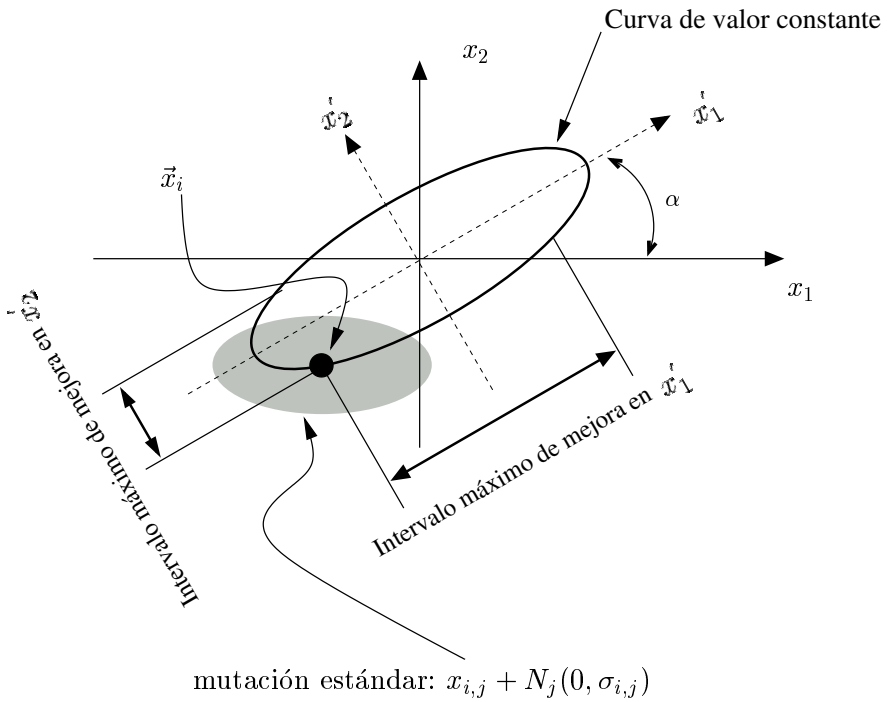


Figura 4.4: Ejemplo de intervalos de mejora usando una función de distribución normal estándar

$\vec{x}_i$ con $\vec{x}_{r_3}$. En una población de np individuos se tienen $np-1$ posibles ejes de búsqueda. El individuo $\vec{x}_{r_3}$ (elegido de manera aleatoria) determina el eje donde se realiza la búsqueda, mientras que el valor de λ determina el punto que será generado a lo largo del eje. Un valor λ diferente a cero introduce un sesgo en la cruza. Cuando $0 < \lambda \leq 1$ las nuevas soluciones se desplazan de $\vec{x}_i$ en dirección a $\vec{x}_{r_3}$ y se alejan tanto de $\vec{x}_i$ como de $\vec{x}_{r_3}$ cuando $\lambda > 1$ o $\lambda < 0$.

En la ecuación (4.3), la inclusión de la diferencia de dos vectores aleatoriamente seleccionados incrementa el número de posibles ejes de búsqueda a $\frac{(np-1)(np-2)}{2}$. Dado que $\vec{x}_{r_1}$ y $\vec{x}_{r_2}$ son elegidos de manera aleatoria, la diferencia $\vec{x}_{r_1} - \vec{x}_{r_2}$ ocurre con la misma frecuencia que $\vec{x}_{r_2} - \vec{x}_{r_1}$ provocando que la mutación exhiba una estadística media de cero. Además, el valor F escala la magnitud de los tamaños de paso y no genera un sesgo como lo hace el factor λ. Se observa además que $F \cdot (\vec{x}_{r_1} - \vec{x}_{r_2})$ tiene el mismo papel en la ED que $N(0, \sigma_{i,j})$ en las EE; sin embargo, la distribución de diferencias es automáticamente escalada por la distribución de los individuos en la población y F únicamente modifica las magnitudes relativas a diferencia de $\sigma_{i,j}$ que define rangos absolutos [75].

Las posibles direcciones de búsqueda y los posibles tamaños de paso dependen completamente de la disposición de los individuos seleccionados. Incrementar el tamaño de la población o el número de diferencias aumenta la diversidad de posibles movimientos e intensifica la exploración del espacio de búsqueda; sin embargo, la probabilidad de encontrar la dirección correcta de búsqueda se decrementa considerablemente. Así, existe un compromiso entre la exploración del espacio de búsqueda y la probabilidad de mejora. En la práctica, entre más complejo sea un problema, más exploración se necesita. El balance entre el tamaño de la población y el número de diferencias determina la eficiencia del algoritmo [27].

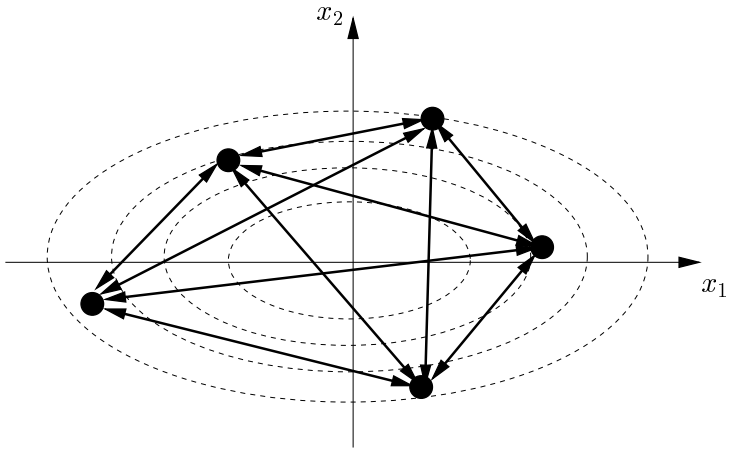


Figura 4.5: Posibles vectores diferencia para el problema del elipsoide

En la figura 4.5 se muestra una población de vectores en el problema del elipsoide. Se observa que la distribución de los vectores diferencia se aproxima a la forma elipsoidal aun cuando no todos los vectores se localizan en la misma curva de nivel. La eficiencia de la distribución de diferencias depende del grado de aproximación de las soluciones al paisaje de aptitud. Es necesario que las soluciones iniciales estén distribuidas uniformemente en todo el espacio de búsqueda para evitar sesgos, además de un proceso de selección determinista que conserve las np mejores soluciones. En cada variable de decisión existen varios tamaños de paso que aproximan a los intervalos máximos de mejora. De la figura se observa que el valor del parámetro F debe ser menor a 1 y debe permanecer constante al menos durante cada generación; el cambiar este factor por cada variable de diseño modifica las posiciones relativas de los individuos y destruye la correlación impuesta por el paisaje de aptitud.

Este esquema de mutación es capaz de escalar los tamaños de paso individualmente así como reducir los efectos en la rotación del sistema coordenado. En la figura 4.6 se ilustra

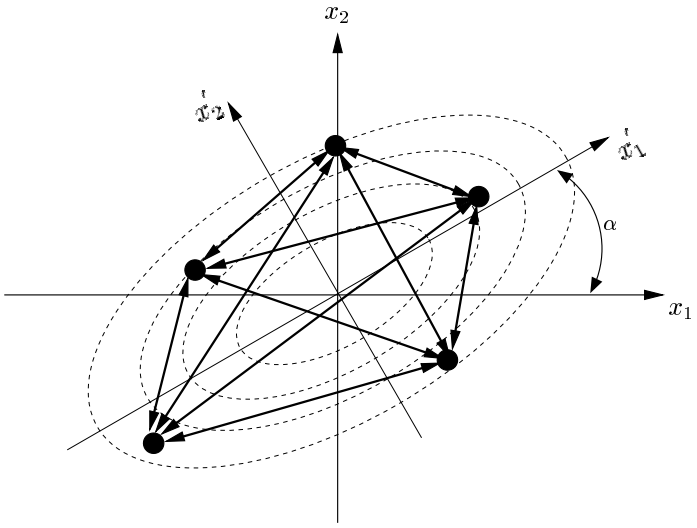


Figura 4.6: Posibles vectores diferencia para el problema del elipsoide con epístasis

el hecho de que una rotación al sistema coordenado no cambia las posiciones relativas de las soluciones con respecto al paisaje de aptitud; así, la distribución de mutación generada tiene la misma orientación que las curvas de nivel del problema.

Otra diferencia importante entre las EE y la ED es la forma en que la información es intercambiada entre individuos. En su forma canónica, las EE no emplean el operador de cruza; así, cada vector solución posee su propio conjunto de valores para escalar y correlacionar la función de distribución multivariable. Estos valores reflejan únicamente la información del vecindario alrededor de cada solución y no existe intercambio global de información. Como consecuencia, se obtienen rápidas mejoras locales pudiendo quedar las soluciones confinadas en un óptimo local.

En contraste, la ED muta todos los vectores con la misma distribución global. En la figura 4.7 se muestra un paisaje de aptitud con dos óptimos locales. Se observa que la distribución de diferencias de la ED contiene una mezcla de vectores diferencia que pertenecen a diferentes óptimos promoviendo así el intercambio de información entre individuos y colaborando con la exploración del paisaje de aptitud.

La mayoría de los métodos clásicos de optimización numérica empleados en la solución de problemas de programación no lineal son de naturaleza iterativa. Estos métodos requieren una solución candidata como punto de inicio; así, proceden a generar una secuencia de soluciones que se aproximen gradualmente al valor óptimo. Las principales diferencias entre

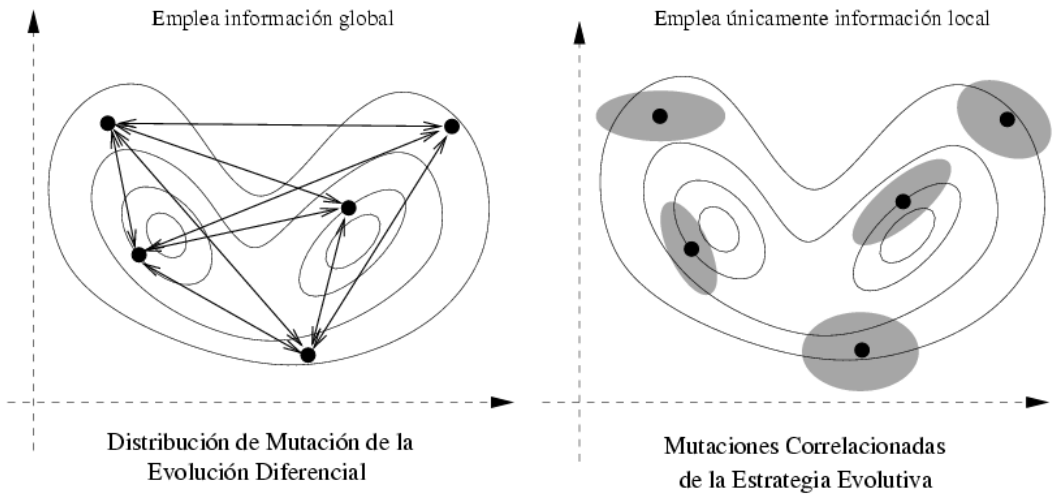


Figura 4.7: Comparación de la distribución de mutaciones de la ED y las mutaciones correlacionadas de la EE

estos métodos radican en la forma de generar las nuevas soluciones así como en la manera de evaluar su optimalidad [76].

Los tres principales métodos de búsqueda directa (métodos que no emplean derivadas de ningún orden de la función objetivo) son: método de Hooke-Jeeves [43], método de Powell [74] y método de Nelder-Mead [69]. Todos ellos emplean el mismo principio: determinar una dirección de búsqueda $\vec{S}$ y generar soluciones en esa dirección; por tal motivo son llamados métodos de búsqueda por patrones [76]. Estos métodos generan vectores de la forma $\vec{Y}_k = \vec{X}_k + \lambda \vec{S}_k$ dada una dirección $\vec{S}_k$ donde $\vec{X}_{k+1} = \vec{Y}_k$ si y sólo si $f(\vec{Y}_k) < f(\vec{X}_k)$ (en el caso de minimización). La generación de vectores de mutación de la ED puede verse como un método de búsqueda por patrones aleatorio basado en poblaciones, donde $\vec{S}_k = \vec{X}_{r_1} - \vec{X}_{r_2}$ y $\lambda = F$. Dada una población de np individuos y m diferencias se obtienen $\prod_{i=1}^{m} (np - i)$ posibles direcciones de búsqueda $\vec{S}_k$ por cada generación.

4.3.2. Recombinación

La mutación es principalmente responsable de generar nuevas direcciones de búsqueda. La recombinación o cruza es un proceso complementario que permite el intercambio de información de los vectores solución a sus descendientes.

En sus primeras versiones [75], la ED emplea un procedimiento de recombinación discreto y no uniforme. En la figura 4.8 se muestra el esquema original de la ED donde los círculos obscuros representan los posibles descendientes ($\vec{u}_i$) generados por el proceso de recombinación entre $\vec{x}_i$ y $\vec{v}_i$.

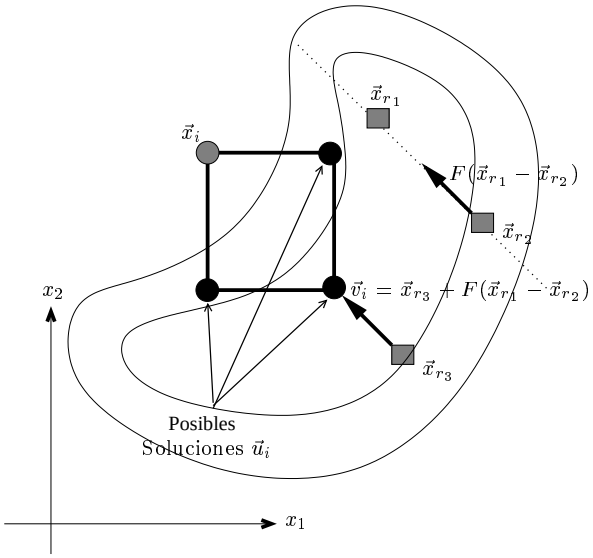


Figura 4.8: Esquema del operador de recombinación del algoritmo de la Evolución Diferencial

Inicialmente se propusieron dos procedimientos de recombinación denominados: exponencial y binomial. Éstos controlan la frecuencia con la cual los parámetros del vector de mutación $\vec{v}_i = \vec{x}_{r_3} + F \cdot (\vec{x}_{r_1} - \vec{x}_{r_2})$ son seleccionados para formar parte de la solución descendiente $\vec{u}_i$. En la figura 4.9 se ilustran ambos procesos, donde $U_i(0,1)$ representa una variable aleatoria uniformemente distribuida en el intervalo $[0,1]$ y el vector $\vec{z}$ es el producto de la recombinación de los vectores $\vec{x}$ y $\vec{y}$. La cruza exponencial es similar a la cruza de un punto [32], mientras que la cruza binomial es similar a la cruza uniforme en los algoritmos genéticos [89].

Francisco Herrera y Manuel Lozano proponen una taxonomía de los principales tipos de recombinación para codificación real además de explicar sus principales características [39]. Esta taxonomía se resume a continuación.

Operadores de cruza discretos Esta categoría agrupa todos los operadores de cruza propuestos para codificación binaria aplicados directamente a la codificación real (p. ej.

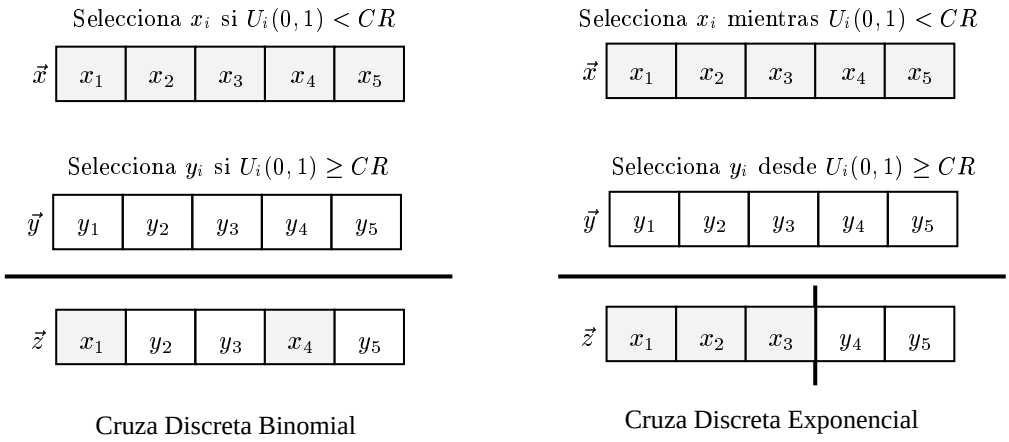


Figura 4.9: Recombinación discreta binomial y exponencial

cruza de un punto, cruza de dos puntos, cruza uniforme, etc.). Estas cruzas muestrean de manera aleatoria las esquinas del hipercubo n-dimensional generado por los vectores padre.

Este tipo de recombinación no es invariante a la rotación y depende de la orientación del sistema coordenado. Además, **no ofrecen diversidad** y únicamente presentan características de **exploración** y no de explotación.

Cruzas basadas en funciones agregativas En esta categoría se incluyen los operadores que usan una función agregativa determinista que numéricamente combina a los padres para generar a los descendientes (p. ej. cruza aritmética, cruza geométrica, cruza heurística).

En estas cruzas, se define la función agregativa $f_i : [\underline{x_i}, \overline{x_i}] \rightarrow [\underline{y_i}, \overline{y_i}]$ dado el intervalo de acción de la i-ésima variable de diseño $[\underline{x_i}, \overline{x_i}]$. f_i tiene propiedades de explotación si $[\underline{y_i}, \overline{y_i}] \subset [\underline{x_i}, \overline{x_i}]$ y/o propiedades de exploración si $[\underline{x_i}, \overline{x_i}] \subset [\underline{y_i}, \overline{y_i}]$; rara vez $[\underline{x_i}, \overline{x_i}] \cap [\underline{y_i}, \overline{y_i}] = \emptyset$.

Cruzas basadas en el vecindario En esta clase de cruzas los descendientes se generan a partir de una cierta distribución de probabilidades de acuerdo a la posición relativa de los padres. Usualmente se emplean más de dos padres y estas cruzas tienen la característica de generar diversidad en la población. Éstas se dividen en dos tipos:

- Cruzas que generan descendientes **cercanos** a los padres (SBX [20], PCX [22]).
- Cruzas que generan descendientes cercanos al **centro geométrico** de los padres (BLX [26], UNDX [71], UNDX-m [51], SPX [91]).

Los esquemas de recombinación binomial y exponencial forman parte de la primer clase de la taxonomía.

Francisco Herrera, Manuel Lozano y Ana María Sánchez elaboraron un estudio donde se examinó la sinergia al emplear más de un operador de recombinación [40]. El estudio consistió en una serie de experimentos donde se combinaron cruzas de las diferentes clases de la taxonomía antes mencionada. Con el fin de medir el rendimiento de las diferentes combinaciones de cruzas se empleó un conjunto de funciones de prueba estándar en la literatura de los algoritmos evolutivos para optimización numérica.

Los esquemas que lograron el mejor rendimiento fueron aquellos donde se combinó una cruza discreta junto con una cruza basada en el vecindario. En su estudio, Francisco Herrera menciona:

> *Es interesante notar que a pesar del pobre desempeño obtenido empleando únicamente la cruza de dos puntos (2P), su sinergia con SBX-2 permite al híbrido resultante (2P & SBX-2) ser uno de los operadores de cruza más competitivos que han sido estudiados en este artículo* [1].

Combinar una cruza con capacidad de explotación (SBX) y una cruza con capacidad de exploración (2P) tiene como resultado un operador con comportamiento de búsqueda robusto. En general, se subestima el rendimiento de las cruzas discretas y se desconoce su capacidad explorativa.

La Evolución Diferencial emplea una distribución global de vectores diferencia como fuente de diversidad; esta distribución posee las características de una cruza basada en el vecindario. Además, la ED emplea un operador de recombinación discreto que ayuda a la exploración del paisaje de aptitud. Así, no es casualidad que los operadores de mutación y recombinación de la ED presenten una sinergia que permite a la ED ser un método de búsqueda robusto.

Por último, cabe mencionar que la distribución de vectores diferencia de la ED posee características que algunas de las cruzas empleadas en el estudio antes mencionado no poseen. Algunas de ellas son: auto-adaptación independiente en los rangos de mutación, invariabilidad a la rotación de los ejes coordenados e intercambio global de información entre los individuos.

[1] Francisco Herrera, Manuel Lozano y Ana María Sánchez [40]

4.3.3. Estructura de la Población

Como se muestra en el algoritmo 5, el cálculo de los individuos de la población de la siguiente generación es mutuamente excluyente; es decir, para implementar de forma paralela a la ED se requiere de un arreglo de individuos que contiene a la población actual y otro arreglo donde se construye la siguiente población; de esta manera se eliminan las dependencias de datos.

Si la ejecución es secuencial, un solo arreglo puede ser usado, incorporando inmediatamente a la población actual los descendientes exitosos. En cualquier caso, no se muestra una diferencia significativa en el rendimiento del algoritmo [75]. Así, una actualización de individuos asíncrona también es posible en una implementación paralela.

4.4. Modelos de Recombinación

Desde sus primeras versiones, la Evolución Diferencial no ha presentado un diseño único. En la literatura se han propuesto diferentes estrategias para los operadores de mutación y recombinación. Por tal motivo, se estableció una nomenclatura que denomina las diferentes versiones de la ED [75]. Por ejemplo, la nomenclatura del algoritmo 5 es **ED/rand/1/bin**, donde *ED* establece que se trata de una versión de la Evolución Diferencial, *rand* indica una elección aleatoria de los vectores que conforman el vector mutación $(\vec{x}_{r_1}, \ldots, \vec{x}_{r_s})$, la cifra 1 señala el número de pares de vectores que conforman la diferencia y *bin* establece un proceso de recombinación **binomial**.

Un algoritmo de ED que selecciona aleatoriamente a cuatro vectores que componen al vector mutación y son recombinados con un proceso **exponencial** se representa como **ED/rand/2/exp**.

Una de las primeras aplicaciones de la ED fue el diseño de filtros digitales donde el modelo **ED/best/2/exp** reveló ser muy efectivo para resolver este problema. En este modelo, *best* señala que la mejor solución en la población es el principal partícipe en el vector mutación $(\vec{v}_i)$, donde $\vec{v}_i = \vec{x}_{best} + \frac{F}{s} \cdot \sum_{k=1}^{p} (\vec{x}_{r_1^k} - \vec{x}_{r_2^k})$ para p diferencias.

La recombinación discreta es un proceso variante a la rotación. Por tal motivo, se han propuesto otros esquemas de recombinación invariantes a la rotación; el más empleado es la recombinación aritmética. La forma del vector descendiente $(\vec{u}_i)$ se muestra en la ecuación (4.5), donde K es el factor λ de la ecuación (4.4) y representa el sesgo de la cruza, cuando $K \to 0$ el vector $\vec{u}_i \to \vec{x}_i$. Por otro lado, cuando $K \to 1$ el vector $\vec{u}_i \to \vec{v}_i$. Este esquema se denomina **DE/current-to-rand/1**, donde *current-to-rand* indica una combinación lineal

de los vectores $\vec{x}_i, \vec{v}_i$.

$$\vec{u}_i = \vec{x}_i + K \cdot (\vec{x}_{r_3} - \vec{x}_i) + F \cdot (\vec{x}_{r_1} - \vec{x}_{r_2}) \tag{4.5}$$

Vitaliy Feoktistov y Stefan Janaqi propusieron una serie de estrategias con el fin de incorporar información de la función objetivo al momento de la mutación / recombinación [27]. Estas estrategias pretender conducir la búsqueda de regiones de baja aptitud a regiones de mejor aptitud.

Tales estrategias consisten en la selección de q individuos, los cuales son divididos en dos conjuntos C_+ y C_- de cardinalidad n_+ y n_- respectivamente ($q = n_+ + n_-$), donde $\forall \vec{x} \in C_+ \wedge \forall \vec{y} \in C_- : f(\vec{x}) < f(\vec{y})$ considerando un problema de minimización. $\vec{X}_{C_\pm}^{max}$ y $\vec{X}_{C_\pm}^{min}$ son los vectores máximos y mínimos respectivamente para cada conjunto.

Los *corrimientos* de cada una de los conjuntos C_+, C_-, se determinan de la siguiente forma.

$$\vec{V}_s^{\pm} = \frac{1}{2} \cdot (\vec{X}_{C_\pm}^{max} - \vec{X}_{C_\pm}^{min})$$

Las nuevas estrategias de diseño son denominadas **rand/n/dir**. La ecuación (4.6) muestra la forma general de estas estrategias, donde $\vec{V}_s = \frac{(\vec{V}_s^+ + \vec{V}_s^+)}{2}$ es el promedio de los vectores de corrimiento y $\vec{V}_{C_+}, \vec{V}_{C_-}$ son los centros geométricos de los vectores que pertenecen a los conjuntos C_+ y C_-, respectivamente.

$$\vec{V}_{C_\pm} = \frac{1}{n_\pm} \sum_{i=1}^{n_\pm} \vec{X}_i$$

$$\vec{v}_i = \vec{V}_{C_+} + F \cdot (\vec{V}_{C_+} - \vec{V}_{C_-} + \vec{V}_s) \tag{4.6}$$

De los resultados experimentales se concluye que el esquema de recombinación **rand/2/dir** muestra una mejora significativa [27]. Cabe mencionar que en estos experimentos no se empleó ningún proceso de recombinación, sino únicamente mutación, esto con el fin de obtener un operador de variación invariante a la rotación.

Finalmente, en la tabla 4.1 se listan los principales modelos de la Evolución Diferencial, propuestos en la literatura.

Nomenclatura	Modelo
rand/p/bin	$u_{i,j} = \begin{cases} x_{r_3,j} + F \cdot \sum_{k=1}^{p} (x_{r_1^p,j} - x_{r_2^p,j}) & \text{si } U_j(0,1) < CR \text{ o } j = j_r \\ x_{i,j} & \text{en caso contrario} \end{cases}$
rand/p/exp	$u_{i,j} = \begin{cases} x_{r_3,j} + F \cdot \sum_{k=1}^{p} (x_{r_1^p,j} - x_{r_2^p,j}) & \text{desde } U_j(0,1) < CR \text{ o } j = j_r \\ x_{i,j} & \text{en caso contrario} \end{cases}$
best/p/bin	$u_{i,j} = \begin{cases} x_{best,j} + F \cdot \sum_{k=1}^{p} (x_{r_1^p,j} - x_{r_2^p,j}) & \text{si } U_j(0,1) < CR \text{ o } j = j_r \\ x_{i,j} & \text{en caso contrario} \end{cases}$
best/p/exp	$u_{i,j} = \begin{cases} x_{best,j} + F \cdot \sum_{k=1}^{p} (x_{r_1^p,j} - x_{r_2^p,j}) & \text{desde } U_j(0,1) < CR \text{ o } j = j_r \\ x_{i,j} & \text{en caso contrario} \end{cases}$
current-to-rand/p	$\vec{u}_i = \vec{x}_i + K \cdot (\vec{x}_{r_3} - \vec{x}_i) + F \cdot \sum_{k=1}^{p} (\vec{x}_{r_1^p} - \vec{x}_{r_2^p})$
current-to-best/p	$\vec{u}_i = \vec{x}_i + K \cdot (\vec{x}_{best} - \vec{x}_i) + F \cdot \sum_{k=1}^{p} (\vec{x}_{r_1^p} - \vec{x}_{r_2^p})$
current-to-rand/p/bin	$u_{i,j} = \begin{cases} \vec{x}_{i,j} + K \cdot (\vec{x}_{r_3,j} - \vec{x}_{i,j}) + F \cdot \sum_{k=1}^{p} (\vec{x}_{r_1^p,j} - \vec{x}_{r_2^p,j}) & \text{si } U_j(0,1) < CR \text{ o } j = j_r \\ x_{i,j} & \text{en caso contrario} \end{cases}$
rand/2/dir	$\vec{v}_i = \vec{v}_1 + \frac{F}{2}(\vec{v}_1 - \vec{v}_2 + \vec{v}_3 - \vec{v}_4)$ donde $f(\vec{v}_1) < f(\vec{v}_2)$ y $f(\vec{v}_3) < f(\vec{v}_4)$

Tabla 4.1: Principales modelos de la Evolución Diferencial ($j_r = U(0,n)$ es un valor aleatoriamente generado en el intervalo $[0,n]$, donde n es el número de variables de decisión)

4.4.1. Estudio Comparativo

Con el fin de ilustrar características y comportamientos reales de los diferentes modelos, se presenta un estudio comparativo empleando los modelos de la Evolución Diferencial listados en la tabla 4.1.

Para evaluar el rendimiento de los modelos seleccionados, se utilizó un conjunto de funciones de prueba estándar en la literatura de los algoritmos evolutivos empleados para la optimización numérica no restringida [92].

El estudio consistió en efectuar 100 ejecuciones independientes por cada modelo mostrado en la tabla 4.1 para cada una de las trece funciones de prueba que se describen en el apéndice A al final de este documento.

Como criterio de terminación se empleó un valor de error absoluto de 10^{-12} con respecto al valor del óptimo global. Se fijó el número máximo de evaluaciones de la función objetivo a **120,000**. Los parámetros empleados en todos los casos son los siguientes.

- Tamaño de la población $\mathbf{NP = 60}$

- Número máximo de generaciones $\mathbf{Gen}_{max} = \mathbf{2000}$

La ED emplea un parámetro de control F que regula las magnitudes relativas de las diferencias del vector mutación. En la literatura, el valor de este parámetro se propone como $F \in [0, 1)$ [75]. Pero, de manera experimental se determinó que el comportamiento del algoritmo no sufre efectos significativos adversos al generar de manera aleatoria el valor de F en el intervalo $[0.3, 0.9]$ durante cada generación [65]. De tal suerte, en esta serie de experimentos se emplea tal propuesta, haciéndose innecesario establecer un valor predeterminado para este parámetro.

En cada uno los modelos de la tabla 4.1 se emplea un parámetro que regula el proceso de recombinación. Cuando se utiliza una recombinación discreta, el parámetro CR especifica una probabilidad; por lo tanto $CR \in [0, 1]$. Cuando se usa una recombinación aritmética, el parámetro K determina el sesgo de la cruza. En algunos casos, valores de $K > 1$ evitan la convergencia del algoritmo; por tal motivo, $K \in [0, 1)$. El valor de este parámetro es dependiente del modelo así como del problema en cuestión. En la mayoría de los casos, el comportamiento del algoritmo se muestra muy susceptible al valor de éste. Por lo tanto, se empleó una búsqueda exhaustiva para determinar el mejor valor de CR o K (según sea el caso) para cada par **modelo-función**.

En todos los casos (exceptuando **rand/2/dir**) únicamente se usó una diferencia en el vector de mutación. Se observó que en la mayoría de los casos emplear un mayor número de diferencias aumenta la velocidad de convergencia del algoritmo pero provoca convergencia prematura. Para evitar esto, es necesario incrementar el tamaño de la población aumentando así el número de evaluaciones. Se advirtió también que se pueden lograr resultados similares con las mismas evaluaciones de la función objetivo utilizando sólo una diferencia con una población menor en vez de emplear dos o más diferencias con un tamaño de población mayor.

También se determinó por cada par **modelo-función** el tamaño mínimo requerido de la población para resolver cada problema dado un número máximo de evaluaciones. El tamaño de la población usado (60) se precisó como el máximo de estos mínimos calculados *(max-min)*.

4.4.2. Problemas de Prueba

Existe un conjunto de funciones de prueba empleados en la optimización numérica sin restricciones usando algoritmos evolutivos. En [92] se listan las veintitrés funciones más empleadas. Trece de estas veintitrés funciones usan 30 variables de decisión y por tal motivo son las empleadas en este estudio. La descripción de estas funciones se encuentra en el Apéndice A al final de este documento.

Estas funciones se clasificaron de acuerdo a si la función es **separable** o no, así como de acuerdo a la modalidad, es decir, el número de óptimos locales: **unimodal** cuando existe un único óptimo local/global o **multimodal** cuando existen múltiples óptimos locales.

En la tabla 4.2 se muestra tal clasificación y además se mencionan algunas características de cada función.

	Modalidad	
Separable	*Unimodal*	*Multimodal*
SI	▪ **f1 (sphere)** función cuadrática simple (esfera n-dimensional) ▪ **f2 (Sch. 2.22)** función discontinua difícil de resolver para métodos clásicos. ▪ **f4 (Sch. 2.21)** Ídem. ▪ **f6** función cuadrática discontinua. ▪ **f7** función polinomial de 4o grado con ruido inicial.	▪ **f8 (Sch. 2.26)** función no polinomial con muchos óptimos locales (infinito número de ellos si el rango no es acotado). El óptimo global cambia según el rango de las variables. ▪ **f9 (Rastrigin)** función no polinomial.
NO	▪ **f3 (Sch. Doble Sum)** función cuadrática difícil de resolver con métodos clásicos (método de Powell, método de Box, método Nelder-Mead).	▪ **f5 (Rosenbrock)** función polinomial de grado 4o. ▪ **f10 (Ackley)** función exponencial continua. ▪ **f11 (Griewangk)** función no polinomial continua. ▪ **f12 (GPF-1)** función irregular y discontinua ▪ **f13 (GPF-2)** función irregular y discontinua

Tabla 4.2: Clasificación de las trece funciones de prueba sin restricciones

Todas estas funciones tienen como valor óptimo a cero a excepción de $f08$. Con la finalidad de mostrar resultados homogéneos, la descripción de $f08$ se ajusta para tener un valor óptimo también de cero.

4.4.3. Resultados

	Funciones				
Modelo	*f01*	*f02*	*f04*	*f06*	*f07*
rand/1/bin	**0.0**	**0.0**	1.9521	**0.0**	**0.0**
rand/1/exp	**0.0**	**0.0**	3.7584	0.843360	**0.0**
best/1/bin	**0.0**	**0.0**	0.0017	**0.0**	**0.0**
best/1/exp	407.972	3.291	1.701872	2737.8458	0.070545
cur_to_best/1	0.54148	4.842	4.233736	1.394	**0.0**
cur_to_rand/1	0.69966	3.503	3.298563	1.767	**0.0**
cur_to_rand/1/bin	**0.0**	**0.0**	0.149514	**0.0**	**0.0**
rand/2/dir	**0.0**	**0.0**	0.044199	**0.0**	**0.0**

Tabla 4.3: Comparación de la aptitud media de 100 ejecuciones independientes de cada modelo analizado para las funciones unimodales y separables $f01$, $f02$, $f04$, $f06$, $f07$. Un resultado en **negritas** indica el valor óptimo global.

En la tabla 4.3 se muestran los resultados promedio de 100 ejecuciones independientes por cada modelo para las funciones unimodales y separables: $f01$, $f02$, $f04$, $f06$, $f07$ (las más comunes de resolver en la literatura). Un resultado remarcado en **negritas** indica que se obtuvo el valor óptimo con un error de 10^{-12}. De esta tabla se observa que los modelos **rand/1/exp, current-to-best/1, current-to-rand/1 y best/1/exp** muestran las peores soluciones. En particular, este último ofrece resultados muy pobres, sobre todo para las funciones $f01$ y $f06$. Por otro lado, los demás modelos exhiben buenos resultados en estas funciones.

	Funciones		
Modelo	*f03*	*f08*	*f09*
rand/1/bin	0.024305	**0.0**	**0.0**
rand/1/exp	0.000004	3.124056	97.753938
best/1/bin	**0.0**	**0.0**	**0.0**
best/1/exp	10.607806	2114.961277	40.003971
cur_to_best/1	0.471730	6240.387323	98.205432
cur_to_rand/1	0.903563	6484.055540	92.263070
cur_to_rand/1/bin	0.000232	0.000001	**0.0**
rand/2/dir	30.112881	**0.0**	**0.0**

Tabla 4.4: Comparación de la aptitud media de 100 ejecuciones independientes de cada modelo analizado para las funciones: $f03$ unimodal y no separable, $f08, f09$ multimodales y separables. Un resultado en **negritas** indica el valor óptimo global.

En la tabla 4.4 se muestran los resultados promedio de 100 ejecuciones independientes por cada modelo para las funciones: $f03$ es unimodal y no separable. $f08$ y $f09$ son mul-

timodales y separables. La función $f08$ es una de las más difíciles de resolver debido a su muy alta modalidad. De esta tabla se observa una vez más que los modelos **rand/1/exp, current-to-best/1, current-to-rand/1 y best/1/exp** exhiben un muy pobre rendimiento. En especial, en las funciones $f08$ y $f09$ se reportan resultados extremadamente pobres. Los demás modelos muestran muy buen rendimiento para $f08$ y $f09$. Únicamente el modelo **rand/2/dir** obtiene un mal desempeño en la función $f03$.

	Funciones				
Modelo	*f05*	*f10*	*f11*	*f12*	*f13*
rand/1/bin	19.577895	**0.0**	0.001117	**0.0**	**0.0**
rand/1/exp	6.696064	0.080037	0.000075	**0.0**	**0.0**
best/1/bin	30.390870	**0.0**	0.000722	**0.0**	0.000226
best/1/exp	132621.5	9.3961	5.9278	1293.0262	2584.85
cur_to_best/1	30.984666	0.270788	0.219391	0.891301	0.038622
cur_to_rand/1	31.702063	0.164786	0.184920	0.464829	5.169196
cur_to_rand/1/bin	24.260535	**0.0**	**0.0**	0.001007	0.000114
rand/2/dir	30.654916	**0.0**	**0.0**	**0.0**	**0.0**

Tabla 4.5: Comparación de la aptitud media de 100 ejecuciones independientes de cada modelo analizado para las funciones multimodales y separables $f05$, $f10$, $f11$, $f12$, $f13$. Un resultado en **negritas** indica el valor óptimo global.

Por último, en la tabla 4.5 se muestran los resultados promedio de 100 ejecuciones independientes por cada modelo para las funciones multimodales y no separables: $f05$, $f10$, $f11$, $f12$, $f13$ (las más difíciles de resolver). Se obtienen muy malos resultados con los modelos **current-to-best/1, current-to-rand/1 y best/1/exp**, mostrando este último resultados extremadamente pobres para las funciones $f05$, $f12$ y $f13$. Los demás modelos muestran un comportamiento competitivo. Cabe mencionar que ninguno de los modelos fue capaz de resolver la función $f05$ (la más difícil) usando $120,000$ evaluaciones de la función objetivo, siendo **rand/1/exp** el que logró el mejor resultado.

De los resultados mostrados en las tablas 4.3, 4.4 y 4.5 se concluye que los modelos **rand/1/exp, best/1/exp, current-to-best/1 y current-to-rand/1** exhiben un rendimiento consistentemente pobre. Es claro que los modelos **current-to-best/1 y current-to-rand/1**, que emplean una recombinación aritmética, poseen una pobre capacidad explorativa que se evidencia al resolver las funciones multimodales, ofreciendo su mejor desempeño en funciones unimodales y separables. El emplear una cruza aritmética destruye la sinergia entre la recombinación discreta y la mutación de vectores diferencia, lo cual concuerda con los resultados obtenidos por Francisco Herrera [40]. Por lo tanto, se obtiene un rendimiento pobre utilizando únicamente el operador de mutación de la ED. Por otro lado, se observa también que los modelos que emplean un proceso de recombinación discreto

exponencial (**rand/1/exp, best/1/exp**) muestran un menor desempeño que sus contrapartes binomiales, en especial **best/1/exp** que mostró ser la peor propuesta de todas. El proceso de recombinación exponencial presenta el problema de que no todas las esquinas del hipercubo n-dimensional formado por las dos soluciones *padre* pueden ser muestreadas sin importar el valor de CR adoptado. De hecho, empleando el proceso exponencial solamente $n+1$ soluciones pueden ser generadas de las 2^n posibles soluciones totales (el proceso binomial puede generar todas estas soluciones, pero no con la misma probabilidad). Esta diferencia de la capacidad explorativa se hace evidente conforme n crece. Tal diferencia es notoria en estos resultados donde $n = 30$.

Con el fin de predecir el comportamiento promedio de los modelos más competitivos (**rand/1/bin, best/1/bin, cur_to_rand/1/bin, rand/2/dir**) se realizó una prueba estadística para calcular los intervalos de confianza tomando como estimador al valor medio. Por cada par **modelo-función** se ejecutó un proceso de *bootstrap* empleando 1000 remuestreos de una muestra de 100 ejecuciones independientes. Los intervalos de confianza con un 95 % de confiabilidad son mostrados en la tabla 4.6.

	Modelos			
Prob.	*rand/1/bin*	*best/1/bin*	*cur_to_rand/1/bin*	*rand/2/dir*
f01	(**0.0**,**0.0**)	(**0.0**,**0.0**)	(**0.0**,**0.0**)	(**0.0**,**0.0**)
f02	(**0.0**,**0.0**)	(**0.0**,**0.0**)	(**0.0**,**0.0**)	(**0.0**,**0.0**)
f04	(1.487,2.453)	(0.0011,0.0024)	(0.1037,0.2053)	(0.0116,0.0874)
f06	(**0.0**,**0.0**)	(**0.0**,**0.0**)	(**0.0**,**0.0**)	(**0.0**,**0.0**)
f07	(**0.0**,**0.0**)	(**0.0**,**0.0**)	(**0.0**,**0.0**)	(**0.0**,**0.0**)
f08	(**0.0**,**0.0**)	(**0.0**,**0.0**)	(**0.0**,**0.0**)	(**0.0**,**0.0**)
f09	(**0.0**,**0.0**)	(**0.0**,**0.0**)	(**0.0**,**0.0**)	(**0.0**,**0.0**)
f03	(0.0206,0.0282)	(**0.0**,**0.0**)	(0.0001,0.0003)	(24.394,37.523)
f05	(16.594,22.773)	(26.399,34.643)	(24.115,24.398)	(28.278,33.651)
f10	(**0.0**,**0.0**)	(**0.0**,**0.0**)	(**0.0**,**0.0**)	(**0.0**,**0.0**)
f11	(0.0005,0.0017)	(0.0002,0.0013)	(**0.0**,**0.0**)	(**0.0**,**0.0**)
f12	(**0.0**,**0.0**)	(**0.0**,**0.0**)	(**0.0**,0.0031)	(**0.0**,**0.0**)
f13	(**0.0**,**0.0**)	(**0.0**,0.0005)	(**0.0**,0.0003)	(**0.0**,**0.0**)

Tabla 4.6: Intervalos de confianza del 95 % para los modelos: *rand/1/bin*, *best/1/bin*, *cur_to_rand/1/bin*, *rand/2/dir* en cada una de las trece funciones de prueba. Estos intervalos se generaron con un procedimiento de *bootstrap* de 1000 re-muestreos para un conjunto de 100 ejecuciones independientes, empleando como estimador al valor medio. Un resultado en **negritas** indica el valor óptimo global.

Todos los modelos muestran un buen comportamiento al resolver las funciones unimodales y separables, aún cuando ninguno resuelve la función $f04$. En la función $f03$ el modelo **rand/2/dir** es el único que muestra un mal desempeño. Para el caso de las funciones mul-

timodales, estos modelos tienen un muy buen desempeño aún cuando ninguno resuelve la función $f05$ de manera satisfactoria. Estos modelos logran resolverla consistentemente incrementando el número de evaluaciones de la función objetivo. De la tabla se muestra que **rand/1/bin, best/1/bin, current-to-rand/1/bin** exhiben comportamientos semejantes.

	Modelos			
Prob.	*rand/1/bin*	*best/1/bin*	*cur_to_rand/1/bin*	*rand/2/dir*
f01	68 %	**30 %***	42 %	48 %
f02	100 %	**35 %***	64 %	70 %
f04	100 %	100 %	100 %	**98 %***
f06	23 %	**12 %***	**12 %***	27 %
f07	40 %	**18 %***	36 %	39 %
f08	100 %	100 %	100 %	100 %
f09	91 %	**78 %***	100 %	90 %
f03	100 %	**99 %***	100 %	100 %
f05	100 %	100 %	100 %	**86 %***
f10	100 %	99 %	**82 %***	100 %
f11	68 %	51 %	**43 %***	50 %
f12	76 %	45 %	**38 %***	54 %
f13	73 %	**37 %***	39 %	56 %

Tabla 4.7: Comparación de los porcentajes del número de evaluaciones de la función objetivo para los modelos: *rand/1/bin*, *best/1/bin*, *cur_to_rand/1/bin*, *rand/2/dir* en cada una de las trece funciones de prueba; se estableció un número máximo de evaluaciones de **120,000** así como una precisión de 10^{-16}. En cada función, el menor porcentaje está indicado por *.

Por último, en la tabla 4.7 se muestra el porcentaje promedio de las $120,000$ evaluaciones de la función objetivo necesarias para resolver cada una de las trece funciones para los modelos **rand/1/bin, best/1/bin, current_to_rand/1/bin, rand/2/dir**. Comparando el modelo **current_to_rand/1/bin** contra **rand/1/bin**, los cuales muestran comportamientos similares, se observa una ligera reducción del número de evaluaciones de la función objetivo. Se obtiene así un efecto positivo al emplear simultáneamente una cruza aritmética y una recombinación discreta. Por otro lado, el modelo **rand/2/dir** no mejora a ninguno de los otros modelos en la mayoría de las funciones (exceptuando $f04$). Aún cuando emplea el menor número de evaluaciones para la función $f05$, es precisamente esta función donde muestra el peor resultado. Esto indica que el modelo perdió diversidad durante el proceso de búsqueda, generando convergencia prematura a consecuencia del sesgo introducido al incorporar información de la función objetivo en el proceso de mutación. Cabe mencionar que esta estrategia mostró su peor desempeño si no se emplea junto con una recombinación discreta, lo cual contradice lo propuesto por Vitaliy Feoktistov y Stefan Janaqi [27].

Finalmente, el modelo **best/1/bin** reveló un comportamiento consistentemente bueno en términos de resultados y de número de evaluaciones de la función objetivo.

4.5. Resumen del Capítulo

La Evolución Diferencial es un método de búsqueda directo empleado en la optimización numérica sin restricciones que forma parte de los algoritmos evolutivos.

El algoritmo utiliza un esquema de mutación invariante a la rotación que auto-adapta los rangos de mutación por cada variable de decisión. Este esquema de mutación no requiere de una función predefinida de probabilidades para generar las perturbaciones.

Otro elemento indispensable en la Evolución Diferencial, como lo muestran los experimentos, es un esquema de recombinación discreto, el cual añade una capacidad explorativa al algoritmo permitiéndole resolver funciones con alta modalidad.

Existen muchos modelos de recombinación/mutación propuestos en la literatura, siendo los modelos **DE/rand/1/bin** y **DE/best/1/bin** aquellos que muestran los mejores rendimientos en términos de resultados y de número de evaluaciones de la función objetivo.

Capítulo 5

Algoritmos Evolutivos para Optimización Restringida

En este capítulo se presenta una introducción de las técnicas más empleadas en los algoritmos evolutivos para manejo de restricciones. Además, en la parte final del capítulo, se presenta una descripción detallada de las propuestas más representativas en el estado del arte para optimización numérica restringida empleado algoritmos evolutivos. La descripción incluye a la técnica más competitiva de Evolución Diferencial existente a la fecha para espacios restringidos.

5.1. Introducción

Las técnicas de programación matemática utilizadas en la solución de problemas de optimización restringida poseen varias limitaciones cuando los planteamientos a resolver poseen algunas de las siguientes características: la función objetivo y/o las restricciones son no diferenciables o discontinuas, la región factible es disjunta o la función objetivo y/o las restricciones son inexpresables de manera algebraica (p. ej. resultado de una simulación). De hecho, las condiciones de Kunh-Tucker para optimalidad no son válidas en el caso general de optimización no lineal; tales condiciones establecen estrictos requisitos que limitan su rango de aplicación.

Los Algoritmos Evolutivos (AE) han sido utilizados exitosamente en la solución de una amplia variedad de problemas de optimización los cuales presentan características como las antes mencionadas. Sin embargo, los AEs en su forma canónica son técnicas de búsqueda sin restricciones. De tal suerte, resulta atractivo incorporar esquemas de manejo de restricciones en los AEs. Hoy en día ésta es una área abierta de investigación.

Coello [6] propone una taxonomía para las principales técnicas de manejo de restricciones en los Algoritmo Evolutivos la cual, se lista a continuación.

- Funciones de Penalización
- Operadores y Representaciones Especiales
- Separación de Objetivos y Restricciones
- Métodos Híbridos

En lo restante de este capítulo se hará una breve introducción a cada una de las clases antes mencionadas.

5.2. Funciones de Penalización

La propuesta de manejo de restricciones más ampliamente usada en los AEs es el uso de funciones de penalización [6]. La idea de este método es transformar un problema de optimización restringido en uno sin restricciones, añadiendo o restando un cierto valor a la función objetivo. Tal valor depende directamente de la violación de restricción para cada solución candidata dada.

Existen dos clases principales de penalización [6]:

Penalización Exterior En este caso, se permiten soluciones infactibles al inicio del proceso de búsqueda, guiando el comportamiento del algoritmo hacia las regiones factibles del espacio de búsqueda.

Penalización Interior En este caso, únicamente se permiten soluciones factibles al inicio del proceso de búsqueda. El factor de penalización extingue a las nuevas soluciones que son infactibles, esto con el fin de guiar la búsqueda únicamente en el interior de la zona factible.

En muchos de los casos, generar una solución factible es un problema difícil. Por tal motivo, en los AEs se han empleado principalmente esquemas de penalización exterior.

La forma general de una función de penalización es la siguiente:

$$\phi(\vec{x}) = f(\vec{x}) \pm \Big[\sum_{i=1}^{m} r_i \cdot G_i + \sum_{j=1}^{p} c_j \cdot L_j\Big] \tag{5.1}$$

Donde $\phi(\vec{x})$ es la función objetivo extendida a optimizar, G_i para $i = \{1, \cdots, m\}$ y L_j para $j = \{1, \cdots, p\}$ son funciones de las restricciones $g_i(\vec{x})$ y $h_j(\vec{x})$ respectivamente, y r_i, c_j son constantes positivas llamadas *factores de penalización*, los cuales determinan la severidad de la penalización.

La formas más comunes de G_i y de L_j son:

$$G_i = max[0, g_i(\vec{x})]^{\beta} \tag{5.2}$$

$$L_j = |h_j(\vec{x})|^{\gamma} \tag{5.3}$$

Donde, normalmente β y γ son uno o dos. Cabe mencionar que la mayoría de los mecanismos de manejo de restricciones empleados en los AEs abordan únicamente problemas con restricciones de desigualdad. En la presencia de restricciones de igualdad, éstas son transformadas en restricciones de desigualdad de la siguiente forma:

$$|h_j(\vec{x})| - \epsilon \leq 0 \tag{5.4}$$

Donde ϵ establece un valor de tolerancia.

Si la penalización es muy alta o muy baja, entonces el problema puede llegar a ser muy difícil de resolver por un AE. Una penalización alta reduce la exploración de la región infactible, mientras que una penalización baja evita una correcta explotación de la región factible. Se sabe que la relación entre un individuo infactible y la región factible del espacio de búsqueda juegan un papel importante en la penalización del individuo. Por ejemplo, Richardson et al. [79] establecen que las penalizaciones que toman en cuenta la distancia a la zona factible presentan mejor desempeño que aquellas que no lo hacen. Sin embargo, no está claro como explotar esta relación *individuo infactible* - *zona factible* para guiar la búsqueda en la dirección correcta [6].

Así, el principal problema de las funciones de penalización es la necesidad de ajustar los coeficientes de penalización r_i, c_j con el fin de lograr el funcionamiento correcto del algoritmo. Estos valores no se conocen *a priori* para un problema arbitrario dado y en general, encontrar tales valores es un problema de optimización difícil por sí mismo.

Se han propuesto diferentes esquemas de penalización en la literatura. A continuación se explican brevemente las principales clases que engloban a estos esquemas.

5.2.1. Pena de Muerte

La pena de muerte consiste en el rechazo total de las soluciones infactibles y es la forma más simple de manejar restricciones. Sin embargo, no es recomendable en el caso general debido a que la búsqueda puede quedar *estancada* al no poder generar soluciones factibles cuando la zona factible es pequeña. El no emplear información de las soluciones infactibles trae como consecuencia un rendimiento pobre del algoritmo, y puede degradar a una simple búsqueda aleatoria.

5.2.2. Penalizaciones Estáticas

En esta categoría se consideran las propuestas donde los factores de penalización no cambian a consecuencia del paso de las generaciones del proceso evolutivo [42, 53]. Sin embargo, no es buena idea mantener los mismos factores de penalización a lo largo de todo el proceso evolutivo. Además, los factores son, en general, dependientes del problema. En algunos casos, el usuario debe definir un número alto de coeficientes de penalización.

5.2.3. Penalizaciones Dinámicas

En esta categoría se incluyen las funciones de penalización que ajustan los factores de penalización conforme el paso de las generaciones del proceso evolutivo [47, 46]. Normalmente los factores son definidos de tal manera que se incrementan conforme pasa el tiempo de evolución. Algunos investigadores han aseverado que las penalizaciones dinámicas funcionan mejor que sus contrapartes estáticas. En general, los problemas asociados con las penalizaciones estáticas también aplican a las penalizaciones dinámicas: si un factor de penalización es incorrecto entonces el AE puede converger prematuramente o simplemente no generar soluciones factibles [6].

5.2.4. Penalizaciones Adaptativas

En este caso se refiere a los métodos que usan información generada por el proceso de búsqueda. Esta clase de propuestas regulan en una forma más *inteligente* la forma de los factores de penalización [34, 14, 15]. Un aspecto interesante de estos métodos es que tratan de mantener una combinación de soluciones factibles e infactibles en la población [6].

5.3. Operadores y Representaciones Especiales

Algunos investigadores han decidido desarrollar esquemas de representación que abordan un tipo de problema en particular, en los cuales un esquema de representación genérico

puede no ser apropiado. El cambio de la representación trae como consecuencia la necesidad de diseñar operadores genéticos especiales que trabajen de una forma similar que los operadores tradicionales.

La principal aplicación de este enfoque es en problemas donde es extremadamente difícil localizar una sola solución factible, o en problemas donde las codificaciones tradicionales muestran un desempeño pobre.

Lawrence Davis [18] emplea varias representaciones y operadores especiales para resolver problemas del mundo real, como generar trayectorias de robots, optimización de horarios, síntesis de la arquitectura de redes neuronales y análisis de ADN.

James C. Bean [70] propuso una representación especial llamada *codificación de llaves aleatorias* la cual es usada para eliminar la necesidad de operadores especiales de recombinación en problemas combinatorios, pues permite representar permutaciones mediante números reales. Esta fue aplicada a una serie de problemas de optimización, por ejemplo: la calendarización de tareas en máquinas paralelas.

Michalewicz [63] propuso al *Algoritmo genético para optimización numérica de problemas restringidos* (**GENOCOP** por sus siglas en inglés). Este algoritmo comienza con una solución factible, y los operadores especiales ejecutan combinaciones lineales de los individuos con el fin de que los descendientes sean factibles también. La principal desventaja de este método es la necesidad de una solución inicial factible, lo cual en muchos de los problemas no es posible. Además, únicamente maneja restricciones lineales y espacios convexos con lo que carece de utilidad, puesto que existen técnicas de programación matemática que resuelven estos problemas de manera exacta para este tipo de problemas.

5.4. Separación de Restricciones y Objetivos

Existen varias propuestas que manejan las restricciones y la función objetivo de manera independiente, es decir no se combina el grado de violación de restricciones y el valor de la función objetivo. Paredis [73] propuso una técnica basada en un modelo co-evolutivo en la cual existen dos poblaciones: la primera contiene las restricciones a ser satisfechas y la segunda contiene soluciones potenciales (posiblemente infactibles). Al final, Paredis [73] indicó que su aproximación es similar a una función de penalización auto-adaptada.

Jiménez et. al y Deb et. al [44, 19] han propuesto de manera independiente el emplear una selección tipo torneo binario utilizando las siguientes reglas de comparación:

- Si las dos soluciones son factibles, aquella con mejor valor de aptitud gana.
- Si una solución es infactible y la otra factible, la solución factible gana.
- Si ambas soluciones son infactibles, aquella con menor grado de violación de restricciones gana.

Ésta es una de las técnicas de manejo de restricciones que no emplean funciones de penalización de manera explícita (no ajusta factores de penalización) más ampliamente usadas en los algoritmos evolutivos. Al final, este torneo binario es equivalente a una función de penalización con un único coeficiente de penalización de muy alto valor, es decir en la ecuación (5.1) $r_1 >> g_i(\vec{x})\ \forall i \in \{1, \ldots, m\}$ y $r_1 >> f(\vec{x})$ para $\forall \vec{x} \in \mathcal{S}$.

También hay varios enfoques que utilizan técnicas de optimización multiobjetivo. Ya que una restricción puede tratarse como un objetivo en el que hay que minimizar el grado de violación, podemos agregar la función objetivo al conjunto de restricciones, siendo todas éstas los objetivos por optimizar [88, 87, 10, 11, 9, 8, 61, 62, 38, 59].

5.5. Métodos Híbridos

En esta categoría se consideran los métodos que se acoplan con alguna otra técnica para manejar restricciones. Adeli y Cheng [1] propusieron un AE que utiliza el método de minimización de Lagrange combinándolo con una función de penalización. Kim y Myung [49] propusieron otro algoritmo evolutivo que emplea una función extendida de Lagrange que garantiza la generación de soluciones factibles durante el proceso de búsqueda. T. Van Lee [57] propuso emplear una combinación de lógica difusa junto con programación evolutiva donde se reemplazan las restricciones originales por restricciones difusas, con el propósito de aumentar el grado de tolerancia en la violación de las restricciones.

5.6. Otras Técnicas

Existen además otras técnicas para el manejo de restricciones, algunas de los cuales emplean otras nuevas heurísticas como el Sistema Inmune Artificial [35, 36, 12, 16], los algoritmos Culturales [78, 45, 7, 56], la optimización de Colonia de Hormigas [82] así como otras técnicas de Manejo de Restricciones, como el método "Adaptive Segregational Constraint Handling" propuesto por Sana Ben Hamida y Marc Schoenauer[37], y los Mapas Homomorfos propuesto por Koziel y Michalewicz [52]. Sin embargo, las Jerarquías Estocásticas propuestas por Runarsson y Yao [80], son sin lugar a dudas la técnica más competitiva de la actualidad de manejo de restricciones en optimización con restricciones.

En las secciones siguientes se muestra una descripción detallada de los tres algoritmos principales en esta tesis (los algoritmos que sirven como base en la comparación del rendimiento de la propuesta de tesis):

- Las Jerarquías Estocásticas [80] por ser la mejor técnica reportada en la literatura.
- La Estrategia Evolutiva Multimiembro Simple [60] que emplea un mecanismo de diversidad similar a la propuesta en esta tesis (capítulo 6.2). Además, muestra resultados competitivos contra las Jerarquías Estocásticas.
- La Evolución Diferencial Restringida [55] como la técnica de manejo de restricciones más competitiva empleando al algoritmo de Evolución Diferencial.

5.6.1. Jerarquías Estocásticas

De todas las propuestas basadas en funciones de penalización, las Jerarquías Estocásticas son la mejor técnica para el manejo de restricciones. Este método, propuesto por Thomas Runarsson y Xin Yao [80], está compuesto por una estrategia evolutiva multi-miembro que emplea una función de penalización. El aspecto más interesante de la propuesta es que no necesita definir factores de penalización.

La motivación principal de las Jerarquías Estocásticas (JE) es balancear la influencia de la función objetivo y el grado de violación de restricciones al momento de asignar el valor de aptitud a un individuo.

La mayoría de las propuestas basadas en funciones de penalización intentan determinar o calcular los valores adecuados de factores de penalización que induzca el balance correcto entre soluciones factibles e infactibles. Las Jerarquías Estocásticas por otro lado, introducen directamente el ordenamiento deseado de los individuos a través de un procedimiento similar al ordenamiento de burbuja. La comparación entre individuos adyacentes se basa tanto en la función objetivo como en la función de penalización, tal elección se efectúa de manera aleatoria con un probabilidad P_f. Este procedimiento se muestra en el algoritmo 6.

Por lo tanto, las Jerarquías Estocásticas imponen una jerarquización en los individuos dominados tanto por la función objetivo como por la función de penalización, la fracción de individuos dominados por cada función dependen directamente del valor de P_f. Los autores determinaron experimentalmente que se logran los mejores resultados cuando $0.4 < P_f < 0.5$ [80].

En su forma original, las Jerarquías Estocásticas emplean una Estrategia Evolutiva (EE) multi-miembro como motor de búsqueda. La EE utilizada emplea mutaciones generadas con

```
1  P = {x⃗_1, x⃗_2, ..., x⃗_λ} ;
2  for i ← {1, ··· , λ} do
3      for j ← {1, ··· , λ − 1} do
4          if ψ(x⃗_j) = ψ(x⃗_{j+1}) = 0 ∨ U(0,1) < P_f then
5              if f(x⃗_j) > f(x⃗_{j+1}) then
6                  Intercambiar(x⃗_j, x⃗_{j+1}) ;
7          else
8              if ψ(x⃗_j) > ψ(x⃗_{j+1}) then
9                  Intercambiar(x⃗_j, x⃗_{j+1}) ;
10     if No intercambio hecho then
11         Terminar() ;
```

Algoritmo 6: Procedimiento de ordenamiento de las Jerarquías Estocásticas donde $U(0,1)$ es un generador de números aleatorios uniformemente distribuidos en el intervalo $[0,1]$.

una distribución normal de probabilidades. Además, emplea un coeficiente que regula el tamaño de paso por cada variable de decisión en cada individuo. Es importante mencionar que no se emplean coeficientes de mutación correlacionados que regulen los ángulos de rotación de las distribuciones que generan la mutación, además de no emplear ningún mecanismo de cruza.

Los autores presentaron una versión mejorada de su algoritmo [81], denominada "Improved Stochastic Ranking Evolution Strategy" (**ISRES** por sus siglas en ingles). En esta nueva versión los cambios efectuados son en el motor de búsqueda empleado, mientras que el mecanismo de manejo de restricciones permanece igual. La mayor diferencia con la versión original es la utilización de una mutación que *suaviza* la auto adaptación de los tamaños de paso. Además, se incluyó una diferenciación estilo Evolución Diferencial que introduce un sesgo hacia las mejores soluciones en el proceso de búsqueda. Se emplea además una recombinación discreta del tipo Evolución Diferencial, en el caso de que las variables de decisión generadas a partir del proceso de mutación estén fuera del rango permitido. Con estas diferencias se logra un mejor rendimiento del algoritmo.

En el algoritmo 7 se muestra la versión mejorada del algoritmo de Jerarquías Estocásticas. Donde, $U_k[0,1]$ representa una variable aleatoria uniformemente distribuida, $\vec{x}_k$ representa a la k-ésima solución dentro de la población, $\vec{\sigma}_k$ es el vector de coeficientes de mutación para el individuo k, λ es el número de descendientes generados, μ el número de padres seleccionados a partir del proceso de jerarquización y γ, α, τ, τ' son parámetros de control

del algoritmo.

```
1  Inicializar cada individuo con: σ'_k = (x̄_k − x_k)/√n   x'_k = x_k + (x̄_k − x_k)U_k[0,1]
2  while Criterio de Terminación NO satisfecho do
3      Evaluar cada solución x⃗_k: f(x'_k), g_i(x'_k) : i = 1,...,n ;
4      Jerarquizar(x⃗_i : i = 1,...,λ) // Empleando el Algoritmo 6 ;
5      (y⃗_j, σ⃗_{y_j}) ← (x⃗_j, σ⃗_j)   j = 1,...,μ // Copiar los μ mejores individuos ;
6      for k = 1 a λ do
7          i = mod(k − 1, μ) + 1 ;
8          if k < μ then
9              // Aplicar Diferenciación ;
10             σ⃗_k = σ⃗_{y_i} ;
11             x⃗_k = y⃗_i + γ(y⃗_1 − y⃗_{i+1}) ;
12             if ∃x_{k_j} ∈ {x_{k_j},...,x_{k_n}} | x_{k_j} > x̄_j ∨ x_{k_j} < x_j then
13                 x_{k_j} = y_{i_j},   ∀x_{k_j} ∈ {x_{k_j} | x_{k_j} > x̄_j ∨ x_{k_j} < x_j } ;
14         else
15             // Aplicar Mutación ;
16             σ_{k_j} = σ_{y_j} e^{τ'·N[0,1]+τ·N_j[0,1]}, j = 1,...,n ;
17             x⃗_k = y⃗_i + σ⃗_y · N[0,1] ;
18             σ⃗_k = σ⃗_y + α(σ⃗_k − σ⃗_y) ;
```

Algoritmo 7: Versión mejorada del algoritmo de las Jerarquías Estocásticas (ISRES).

Es importante mencionar que la diferenciación y la recombinación discreta son elementos fundamentales en el algoritmo de la Evolución Diferencial. De hecho, la sinergia entre ambos es lo que la caracteriza. No es sorprendente que al emplear tales elementos se mejore de manera significativa el rendimiento del algoritmo.

5.6.2. Estrategia Evolutiva Multimiembro Simple

La hipótesis en la que se basa esta propuesta es la siguiente [60]: (1) El mecanismo de auto-adaptación de una Estrategia Evolutiva permite muestrear el espacio de búsqueda lo suficiente para localizar la región factible razonablemente rápido y (2) la simple adición de un criterio de selección basado en factibilidad guía la búsqueda en tal manera que el óptimo global puede aproximarse eficientemente [60].

El criterio de selección mencionado (basado en comparaciones entre parejas de soluciones) es el siguiente:

- Si las dos soluciones son factibles, aquella con mejor valor de aptitud gana.
- Si una solución es infactible y la otra factible, la solución factible gana.
- Si ambas soluciones son infactibles, aquella con menor grado de violación de restricciones gana.

Este criterio ha sido empleado por otros autores [44, 19]. Sin embargo, la propuesta de Jiménez y Verdegay [44] carece de un mecanismo que evite la convergencia prematura, mientras que Deb [19] emplea una técnica de nichos para evitarla. Empleando el criterio anterior las soluciones factibles son siempre preferidas que las soluciones infactibles; así, estas técnicas tienen dificultades en problemas donde el óptimo global se localiza en el límite de la zona factible [60].

La propuesta de Mezura [60], laEstrategia Evolutiva Multimiembro Simple (**SMES** por sus siglas en inglés), emplea una Estrategia Evolutiva $(\mu + \lambda)$ como motor de búsqueda pero, en este caso, se emplea un mecanismo de diversidad el cual tiene las siguientes características [60]: En cada generación, se permite que el individuo infactible con el mejor valor de aptitud y con el menor grado de violación de restricciones forme parte de la siguiente población. A este individuo se denomina la mejor solución infactible. De hecho, en la población de la siguiente generación se incluye uno de los dos mejores individuos factibles; uno de ellos de los μ padres y el otro de los λ descendientes. Cada uno de ellos se selecciona con la misma probabilidad del 50 %. El proceso de selección se muestra en el algoritmo 8.

La misma solución infactible puede ser copiada más de una vez en la siguiente población. Sin embargo, mantener *pocas* copias de esta solución permite su recombinación con *varias* soluciones en la población, especialmente con soluciones factibles [60]. Recombinar soluciones factibles e infactibles permite encontrar el óptimo global en problemas donde éste se localiza cerca del límite de la región factible [60].

Del algoritmo 8 se observa que el número de soluciones infactibles es relativamente bajo: sólo el 3 %. Sin embargo, según los autores esto es suficiente para proveer resultados competitivos con respecto a otros algoritmos representativos en el área de optimización restringida de los algoritmos evolutivos.

El valor de tolerancia ϵ empleado en la transformación de restricciones de igualdad a restricciones de desigualdad se decrementa con el paso de las generaciones usando la siguiente expresión: $\epsilon(t+1) = \frac{\epsilon(t)}{\beta}$ donde β es un parámetro de control del algoritmo que se ajusta

```
1  // Se calcula la unión de los μ padres y los λ descendientes ;
2  Q = {x⃗_1, ..., x⃗_μ} ∪ {y⃗_1, ..., y⃗_λ} ;
3  // P es la población de la siguiente generación ;
4  P = ∅ ;
5  for i = 1 a μ do
6      if flip(0.97) then
7          Selecciona al mejor individuo q⃗ de Q empleando el criterio de comparación
           basado en factibilidad ;
8          Q = Q − {q⃗} ;
9          P = P ∪ {q⃗} ;
10     else
11         if flip(0.5) then
12             Seleccionar al mejor individuo infactible x⃗_best de los μ padres ;
13             P = P ∪ {x⃗_best} ;
14         else
15             Seleccionar al mejor individuo infactible y⃗_best de los λ descendientes ;
16             P = P ∪ {y⃗_best} ;
```

Algoritmo 8: Pseudo-código del procedimiento de selección de SMES con el mecanismo de diversidad incorporado. $flip(p)$ es una función cuyo valor de retorno es *VERDADERO* con una probabilidad p.

según el problema.

Un aspecto importante de esta propuesta es la utilización de un operador de cruza resultado de una mezcla de una cruza intermedia y una cruza discreta [60]. En el algoritmo 9 se muestra en pseudo-código del procedimiento de recombinación.

```
1  Seleccionar x⃗_1 de la población P ;
2  // Para cada variable de decisión ;
3  for i = 1 a n do
4  |  Seleccionar x⃗_2 de la población P ;
5  |  if flip(0.5) then
6  |  |  if flip(0.5) then
7  |  |  |  y_i = x_{1_i} ;
8  |  |  else
9  |  |  |  y_i = x_{2_i} ;
10 |  else
11 |  |  y_i = x_{1_i} + (x_{2_i} − 1/2 · x_{1_i})
12 regresar a y⃗ como descendiente ;
```

Algoritmo 9: Pseudo-código del procedimiento de recombinación empleado por el algoritmo SMES

Se aprecia del algoritmo 9 que el proceso de cruza es sorprendentemente similar al proceso de recombinación de la Evolución Diferencial. Se emplea una cruza discreta junto con una diferenciación. A pesar de que la diferenciación empleada sólo genera posibles direcciones de búsqueda a lo largo de la línea que une a $\vec{x}_1$ y $\vec{x}_2$, el vector $\vec{x}_2$ cambia por cada variable de decisión, aumentando así las posibles direcciones de búsqueda.

Una vez más, no resulta sorprendente que la incorporación de la diferenciación y la recombinación discreta mejoren sensiblemente el rendimiento del algoritmo.

5.6.3. Evolución Diferencial Restringida

En la literatura [54] de Evolución Diferencial, la mayoría de los métodos de manejo de restricciones empleados son las funciones de penalización. A diferencia de estas propuestas, Lampinen et. al [55] propusieron una modificación al proceso de selección de Evolución Diferencial con la finalidad de manejar restricciones. Lo atractivo de la propuesta, llamada "Constrained Differential Evolution" (**CDE** por sus siglas en inglés), es que evita emplear

funciones de penalización. Además, sólo modifica al proceso de selección, manteniendo la estructura del algoritmo original de la Evolución Diferencial.

$$\vec{x}_i^{(t+1)} = \begin{cases} \vec{u}_i^{(t)} & \text{Si} \begin{cases} \begin{cases} \forall j \in \{1,\ldots,m\}: \ g_j(\vec{u}_i^{(t)}) \leq 0 \ \wedge \ g_j(\vec{x}_i^{(t)}) \leq 0 \\ \wedge \\ f(\vec{u}_i^{(t)}) \leq f(\vec{x}_i^{(t)}) \end{cases} \\ \vee \\ \begin{cases} \forall j \in \{1,\ldots,m\}: \ g_j(\vec{u}_i^{(t)}) \leq 0 \\ \wedge \\ \exists j \in \{1,\ldots,m\}: \ g_j(\vec{x}_i^{(t)}) > 0 \end{cases} \\ \vee \\ \begin{cases} \exists j \in \{1,\ldots,m\}: \ g_j(\vec{u}_i^{(t)}) > 0 \\ \wedge \\ \forall j \in \{1,\ldots,m\}: \ max[g_j(\vec{u}_i^{(t)}),0] \leq max[g_j(\vec{x}_i^{(t)}),0] \end{cases} \end{cases} \\ \vec{x}_i^{(t)} & \text{en otro caso} \end{cases} \tag{5.5}$$

En la ecuación (5.5) se muestra el criterio de comparación propuesto entre el vector descendiente $\vec{u}_i^{(t)}$ y el individuo i-ésimo en la población $\vec{x}_i^{(t)}$ de la generación t, para determinar al individuo i-ésimo de la siguiente generación $\vec{x}_i^{(t+1)}$. En este caso se considera un problema de minimización con m restricciones de desigualdad (se asume que las restricciones de igualdad han sido transformadas).

Empleando el criterio anterior, el vector descendiente $\vec{u}_i^{(t)}$ es seleccionado para la siguiente generación sobre $\vec{x}_i^{(t)}$ si:

- Satisface todas las restricciones y provee un valor de la función objetivo igual o mejor.
- Si $\vec{u}_i^{(t)}$ es factible y $\vec{x}_i^{(t)}$ no lo es.
- Es infactible pero provee de una menor o igual violación de restricción para cada una de las m restricciones.

Se observa de la ecuación (5.5) que el criterio de comparación propuesto por Lampinen et. al [55] es muy similar a los propuestos por Jiménez et. al y Deb et. al [44, 19]:

- Si las dos soluciones son factibles, aquella con mejor valor de aptitud gana.
- Si una solución es infactible y la otra factible, la solución factible gana.

- Si ambas soluciones son infactibles, aquella con menor grado de violación de restricciones gana.

El cambio ocurre en la última regla, en este caso cuando las dos soluciones son infactibles se selecciona a $\vec{u}_i^{(t)}$ si no viola más a ninguna de las restricciones que el vector $\vec{x}_i^{(t)}$. Obsérvese que este criterio es análogo al concepto de optimalidad de Pareto empleado en optimización multiobjetivo.

Al emplear un criterio de óptimo de Pareto no en todos los casos se puede determinar que una solución es mejor que la otra. En tal caso, bajo el concepto de optimalidad de Pareto se dice que son no dominadas entre sí. Por lo tanto, el criterio de la ecuación (5.5) únicamente acepta reemplazo en la población cuando se obtienen mejoras en todas las restricciones. Esto limita mucho su capacidad de exploración, puesto que es muy poco probable que el operador de recombinación genere soluciones que mejoren todas las restricciones, provocando así *estancamiento*. Además, el mecanismo de manejo de restricciones no es capaz de explotar de manera eficiente la frontera de la zona factible puesto que existe una presión de selección muy alta hacia las zonas factibles del espacio de búsqueda. Esto se aprecia en los resultados experimentales mostrados en el [55] y los del capítulo 7 de esta tesis.

La principal desventaja de esta técnica es la no utilización de ningún mecanismo de diversidad entre individuos factibles e infactibles. Además, conforme el número de restricciones aumenta, el número de casos donde las soluciones son no dominadas aumenta también; la propuesta no permite el reemplazo de soluciones no dominadas, evitando *movimientos* entre soluciones equivalentes según el criterio de Pareto. En la práctica, esto trae como consecuencia que el algoritmo CDE requiera de muchas evaluaciones de la función objetivo en la mayoría de los problemas de prueba.

Es importante mencionar que esta propuesta emplea al modelo **rand/1/bin** y no se considera a ningún otro.

5.7. Resumen del Capítulo

Se presentó una taxonomía que cubre a las principales técnicas de manejo de restricciones empleadas en los algoritmos evolutivos. Al final se hace una descripción detallada de tres algoritmos principales:

- Las Jerarquías Estocásticas: La mejor técnica en la literatura.
- La Estrategia Evolutiva Multimiembro Simple: La técnica que emplea un mecanismo de diversidad, que retiene individuos infactibles cercanos a la zona factible.

- La Evolución Diferencial Restringida: La mejor técnica empleando Evolución Diferencial.

Es importante mencionar que tanto las Jerarquías Estocásticas como la Estrategia Evolutiva Multimiembro Simple emplean elementos propios de la Evolución Diferencial tales como una diferenciación entre soluciones y un proceso de recombinación discreto; de hecho la sinergia entre ambos es lo que caracteriza a la ED. Otro elemento importante en estas dos técnicas es que intentan obtener un balance entre las soluciones factibles e infactibles.

Finalmente, cabe mencionar que a pesar de que existen muchas técnicas de manejo de restricciones en los algoritmo evolutivos, las Jerarquías Estocásticas son indiscutiblemente la mejor técnica reportada a la fecha en la literatura especializada.

Capítulo 6

Descripción de la Propuesta

En este capítulo se presenta la descripción completa de la propuesta del algoritmo de Evolución Diferencial la cual consiste en un nuevo modelo de evolución diferencial y un mecanismo de manejo de restricciones.

La primera parte del capítulo trata acerca del nuevo modelo propuesto para optimización global sin restricciones. Después se detalla el mecanismo de manejo de restricciones propuesto y, finalmente, se presenta una descripción completa del algoritmo de evolución diferencial para optimización restringida.

6.1. Nuevo Modelo de Recombinación de la Evolución Diferencial

En el capítulo 4 se hizo mención de los principales lineamientos que componen al algoritmo de la Evolución Diferencial. Al final se concluyó de manera experimental, que el modelo $best/1/bin$ muestra los mejores resultados usando el conjunto de funciones de prueba estándar de la literatura de los algoritmos evolutivos para optimización global sin restricciones. Este modelo no sólo presenta un comportamiento promedio mejor que otras propuestas sino que además emplea un menor número de evaluaciones de la función objetivo. Esto contrasta con el hecho de que el modelo de ED más ampliamente usado es $rand/1/bin$.

La Evolución Diferencial, a diferencia de otros algoritmos evolutivos (p. ej. algoritmos genéticos), provee un mecanismo de recombinación de soluciones basado en combinaciones lineales. Generar combinaciones lineales a partir de un conjunto de soluciones candidatas de referencia puede ser visto como un proceso de generación de trayectorias hacia dentro y fuera del espacio generado por tales soluciones, donde los nuevos vectores sirven como punto de inicio de futuras trayectorias. Esto lleva a una concepción más amplia acerca de la

recombinación de soluciones. Una trayectoria entre soluciones generará nuevos individuos que comparten un subconjunto de atributos provenientes de las soluciones padre.

En la búsqueda dispersa [31], la forma de generar tales trayectorias es comenzando con una solución de referencia inicial, a la cual se aplican diferentes *movimientos* que introducen progresivamente atributos contribuidos por una solución guía. La incorporación parcial o total de atributos de soluciones élite en la generación de nuevos individuos es un aspecto fundamental en la búsqueda dispersa. Así, la recombinación en la búsqueda dispersa emplea estrategias de diseño, mientras que en los algoritmos genéticos se espera que por medio de la aleatoriedad se descubran tales estrategias.

El modelo propuesto se basa en la idea de incorporar información de las soluciones élite a través de una estrategia de búsqueda basada en la aleatoriedad. En la tabla 6.1 se muestra una comparación de los dos principales modelos de la ED. Éstos se componen de un vector *donante* y un vector *diferencia*. El vector donante funciona como punto de inicio de la trayectoria, mientras que el vector diferencia especifica la dirección de búsqueda.

modelo	**Donante**	**Diferencia**
rand/1/bin	$\vec{x}_{r_3}^{(g)}$	$F \cdot (\vec{x}_{r_2}^{(g)} - \vec{x}_{r_1}^{(g)})$
best/1/bin	$\vec{x}_{best}^{(g)}$	$F \cdot (\vec{x}_{r_2}^{(g)} - \vec{x}_{r_1}^{(g)})$

Obsérvese que en ambos casos, los vectores diferencia se generan de la misma forma: seleccionando dos soluciones de manera aleatoria dentro de la población. La única diferencia radica en el punto de inicio de la trayectoria. En la figura 6.1 se muestra un ejemplo de los posibles vectores diferencia para una población de cuatro soluciones. En la figura se ilustra el hecho de que la secuencia de soluciones $\{\vec{x}_i^{(0)}, \ldots, \vec{x}_i^{(g)}\}$ para el vector solución en la posición i dentro de la población a lo largo de g generaciones constituye una trayectoria. Esto último es consecuencia del proceso de selección de la ED que compara únicamente a la solución padre contra la solución descendiente.

La nueva propuesta consiste en incorporar información de la mejor solución de la población en la diferencia; es decir, se generan trayectorias con dirección a la mejor solución utilizando la diferencia $(\vec{x}_{best}^{(g)} - \vec{x}_{r_1}^{(g)})$. En la figura 6.2 se muestran las posibles direcciones de búsqueda empleando como solución guía a la mejor solución de la población.

Se introduce un sesgo muy fuerte en la búsqueda al incorporar únicamente la diferencia $(\vec{x}_{best}^{(g)} - \vec{x}_{r_2}^{(g)})$ en el vector mutación; además, no se utiliza información alguna de la trayectoria de $\vec{x}_i$. Por tales motivos, se introduce además la diferencia $(\vec{x}_i^{(g)} - \vec{x}_{r_1}^{(g)})$ con el fin de

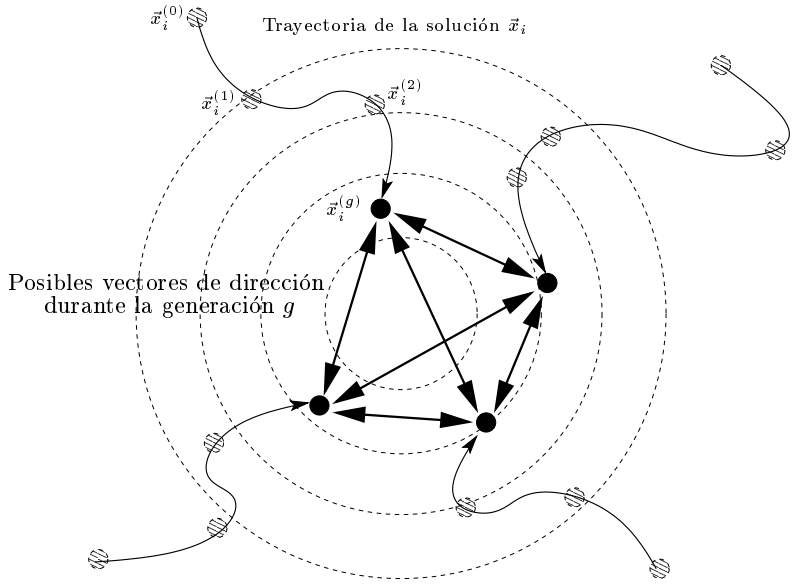


Figura 6.1: Ejemplo de posibles vectores de diferencia para los modelos *rand/1/bin* y *best/1/bin*

incorporar información de la trayectoria. Se emplea a la solución $\vec{x}_i^{(g)}$ ya que ésta representa a la mejor solución dentro de la trayectoria del vector $\vec{x}_i$.

El modelo propuesto para la generación del vector mutación $\vec{x}_{m_i}$ se muestra en la ecuación 6.1, donde los coeficientes F_α, F_β determinan las proporciones que regulan la dirección de búsqueda: ya sea por parte de la mejor solución (información global) o por parte de la mejor solución en la trayectoria de $\vec{x}_i$ (información local) respectivamente. Además, en la mayoría de los casos $F_\beta + F_\alpha < 1.0$. De otra manera, el algoritmo puede diverger.

$$\vec{x}_{m_i} = \vec{x}_{r_3}^{(g)} + F_\alpha \cdot (\vec{x}_{best}^{(g)} - \vec{x}_{r_2}^{(g)}) + F_\beta \cdot (\vec{x}_i^{(g)} - \vec{x}_{r_1}^{(g)}) \tag{6.1}$$

En la figura 6.3 se muestra el proceso de generación del vector de mutación $\vec{x}_{m_i}$. Es importante mencionar que este vector no es la solución descendiente del vector $\vec{x}_i$. Es necesario aplicar un proceso de recombinanción discreto (binomial) entre los vectores $\vec{x}_{m_i}^{(g)}$ y $\vec{x}_i^{(g)}$ con el fin de calcular la solución descendiente $\vec{u}_i^{(g)}$.

Es importante mencionar que existe una diferencia notable entre el modelo *best/1/bin*

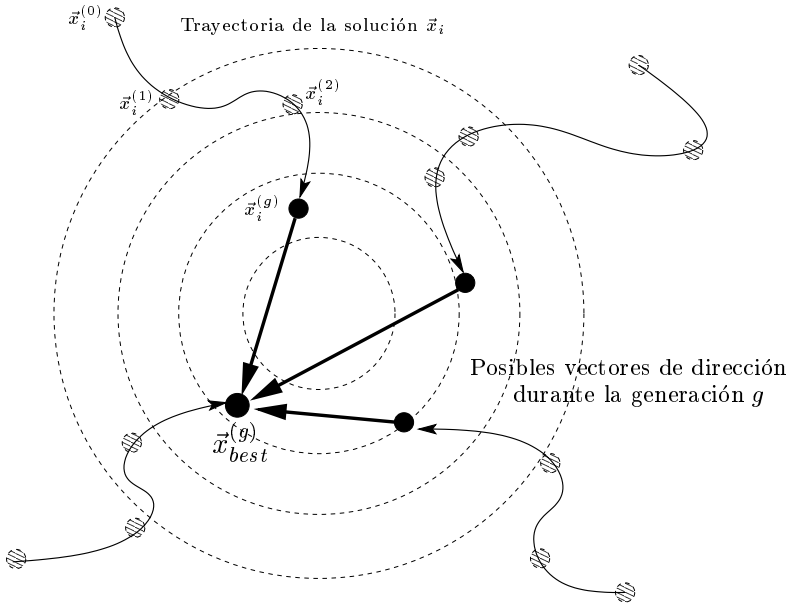


Figura 6.2: Ejemplo de posibles vectores de diferencia considerando a $\vec{x}_{best}^{(g)}$ como vector guía

y la nueva propuesta. Emplear a la mejor solución $\vec{x}_{best}^{(g)}$ como punto inicial de la trayectoria implica que los vectores mutación generados son perturbaciones locales de la mejor solución $\vec{x}_{best}^{(g)}$; mientras que al emplear la diferencia $\vec{x}_{best}^{(g)} - \vec{x}_{r2}^{(g)}$ se concentra el proceso de búsqueda en regiones críticas de alta aptitud, reduciendo las posibles direcciones de búsqueda y guiando a las trayectorias hacia dichas zonas. Emplear únicamente a $\vec{x}_{best}^{(g)}$ como solución guía es insuficiente ya que en ningún momento la historia del vector $\vec{x}_i$ es tomada en cuenta. De ahí la necesidad de incorporar la diferencia $\vec{x}_i^{(g)} - \vec{x}_{r1}^{(g)}$ en el vector mutación, ésto como una forma de incorporar tal información.

Nuevos desarrollos han mostrado que la hibridación de técnicas evolutivas junto con buscadores locales incrementan la eficiencia de la búsqueda [33]. De hecho, los algoritmos genéticos que emplean un método de refinamiento local son catalogados en una clase especial de algoritmos llamados meméticos [67].

Los Algoritmos Meméticos (AM) emplean un proceso aparte de búsqueda local para *mejorar* a los individuos (v.g., incrementar su aptitud). Los AM pretenden proveer métodos de optimización global tomando ventaja de la capacidad de exploración de los algoritmos genéticos y la capacidad de explotación de los buscadores locales (BL). Los AM se clasifican en dos grandes grupos dependiendo del tipo de BL que se emplee; el primer grupo

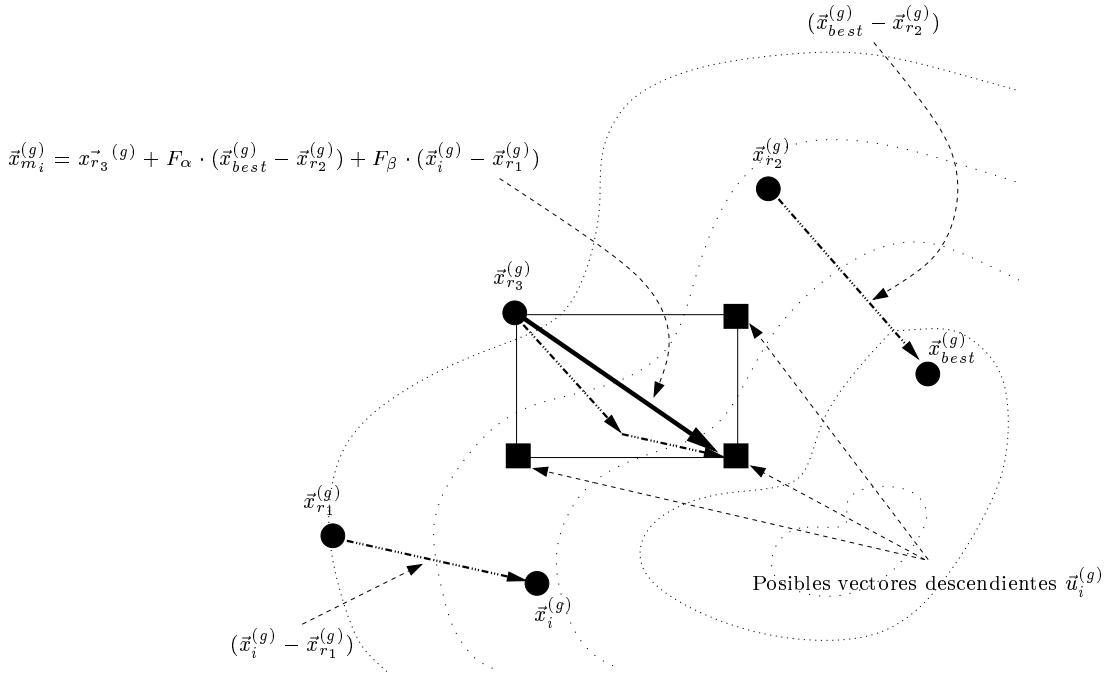


Figura 6.3: Ilustración de la generación de descendientes a partir del vector *padre* $\vec{x}_i^{(t)}$

lo componen los Procesos de Mejoramiento Local (PML) y el segundo está compuesto por Buscadores Locales basados en Cruza (BLX) [58]. En la primer categoría las soluciones son perfeccionadas en cada generación empleando BL del tipo *escalando la colina* (hill-climbers); en el otro grupo, como su nombre lo siguiere, un operador de cruza es empleado para el perfeccionamiento local de las soluciones.

En la literatura se han empleado diferentes tipos de cruzas para parámetros reales que tienen naturaleza auto-adaptativa y que pueden generar descendientes basándose en la distribución de los *padres* sin emplear ningún parámetro de control [3], [50], [22], [71], [91]. Una de las características comunes en muchos de los buscadores locales basados en cruza es la generación de múltiples descendientes a partir de un conjunto de soluciones élite (similar a la búsqueda dispersa [31]). Su motivación principal es generar múltiples descendientes en el vecindario de un conjunto reducido de soluciones de referencia sobre generaciones sucesivas [72].

Como se mencionó en el capítulo 4, el esquema de mutación de la ED es auto-adaptado, invariante a la rotación, comparte información global y no requiere de una función predefinida de probabilidades para generar las perturbaciones. Así, en lugar de emplear un método

de cruza complementario para mejorar la capacidad explorativa del algoritmo, simplemente se generan más de un descendiente por cada una de las soluciones dentro de la población y el mejor de ellos compite contra el *padre* por un lugar en la población de la siguiente generación.

De tal modo, los elementos que constituyen al nuevo modelo de evolución diferencial son:

- Un vector de mutación que emplea la información de la mejor solución en la población y la mejor solución en la trayectoria actual para determinar la dirección de búsqueda, tomando como inicio de la trayectoria a una solución dentro de la población seleccionada de manera aleatoria.
- Un proceso de recombinación discreto binomial cuya sinergia con el vector de mutación incrementa la capacidad explorativa del algoritmo.
- Un buscador local que permite incrementar la capacidad de exploración del algoritmo; el cual consiste en la generación de múltiples descendientes para cada solución *padre* usando el mismo esquema de recombinación.

El algoritmo 10 muestra en pseudo-código la descripción de la propuesta de Evolución Diferencial para optimización global sin restricciones; donde f es la función de aptitud, n es el número de variables de decisión del problema, $\vec{x}_i^{(g)}$ representa al i-ésimo individuo de la población en la generación g, $\vec{x}_{best}^{(g)}$ representa al individuo con mejor aptitud en la población, $x_{i,j}^{(g)}$ es la j-ésima variable de decisión del i-ésimo individuo en la población, $\vec{u}_i$ es el vector resultante del proceso de recombinación basado en el i-ésimo individuo de la población y $U_j[0,1]$ es un valor generado aleatoriamente por cada variable de decisión con una distribución uniforme en el intervalo $[0,1]$.

Los parámetros de control del algoritmo propuesto son los siguientes:

- G_{max}: número máximo de generaciones.
- μ es el número de individuos que componen a la población.
- λ: * número de descendientes generados por cada individuo en la población.
- CR: parámetro de control de la cruza discreta
- F_α: * factor de escalamiento de la dirección de búsqueda impuesta por $\vec{x}_{best}^{(g)}$.
- F_β: * factor de escalamiento de la dirección de búsqueda impuesta por $\vec{x}_i^{(g)}$.

Los parámetros marcados con * no forman parte del algoritmo original de la evolución diferencial. De manera experimental se determinó que un valor de $2 \leq \lambda \leq 7$ es apropiado, así como $F_\alpha = 0.8$ y $F_\beta = 0.1$ se mostraron suficientes para el conjunto de trece funciones de prueba sin restricciones.

El único parámetro que se mostró sensible y dependiente del problema fue el valor CF que controla al operador de cruza discreta.

```
1  Inicializar(P_g ← {x⃗_1^(0), ..., x⃗_μ^(0)})
2  for g ← {1, ..., G_max} do
3      for i ← {1, ..., μ} do
4          x⃗_itmp^(g) = x⃗_i^(g) ;
5          for k ← {1, ..., λ} do
6              r_1, r_2, r_3 ∈ {1, ..., μ} aleatoriamente seleccionados,
7                                   donde r_1 ≠ r_2 ≠ r_3 ≠ i ;
8              j_rand ∈ {1, ..., n} aleatoriamente seleccionado ;
9              for j ← {1, ..., n} do
10                 if U_j[0,1] < CR or j = j_rand then
11                     u_{i,j}^(g) ← x_{r3,j}^(g) + F_α · (x_{best,j}^(g) − x_{r2,j}^(g)) + F_β · (x_{i,j}^(g) − x_{r1,j}^(g)) ;
12                 else
13                     u_{i,j}^(g) ← x_{i,j}^(g) ;
14             if f(u⃗_i^(g)) ≤ f(x⃗_itmp^(g)) then
15                 x⃗_itmp^(g) ← u⃗_i^(g) ;
16         x⃗_i^(g+1) = x⃗_itmp^(g) ;
```

Algoritmo 10: Algoritmo de evolución diferencial propuesto para optimización global sin restricciones *newde* suponiendo un problema de minimización

6.2. Mecanismo de Manejo de Restricciones

Como se ha mencionado en capítulos anteriores, el método más ampliamente usado para el manejo de restricciones, incluyendo a las propuestas basadas en el algoritmo de Evolución Diferencial [75], son las funciones de penalización.

Sin pérdida de generalidad, consideremos únicamente problemas de minimización. Supongamos ahora, a la secuencia $S_i = \{f(\vec{x}_i^{(1)}), f(\vec{x}_i^{(2)}), \ldots, f(\vec{x}_i^{(t)})\}$ de valores de aptitud del i-ésimo vector $\vec{x}_i$ en la población durante t generaciones (trayectoria del vector $\vec{x}_i$), tal como se muestra en la figura 6.4. El algoritmo original de la Evolución Diferencial emplea un mecanismo de selección determinista; esto es, la solución $\vec{u}_i^{(t)}$, generada a partir del vector $\vec{x}_i^{(t)}$ en la generación t, reemplaza a la solución $\vec{x}_i^{(t)}$ en la siguiente generación si y sólo si $f(\vec{u}_i^{(t)}) \leq f(\vec{x}_i^{(t)})$, de tal modo $\vec{x}_i^{(t+1)} = \vec{u}_i^{(t)}$; en caso contrario $\vec{x}_i^{(t+1)} = \vec{x}_i^{(t)}$.

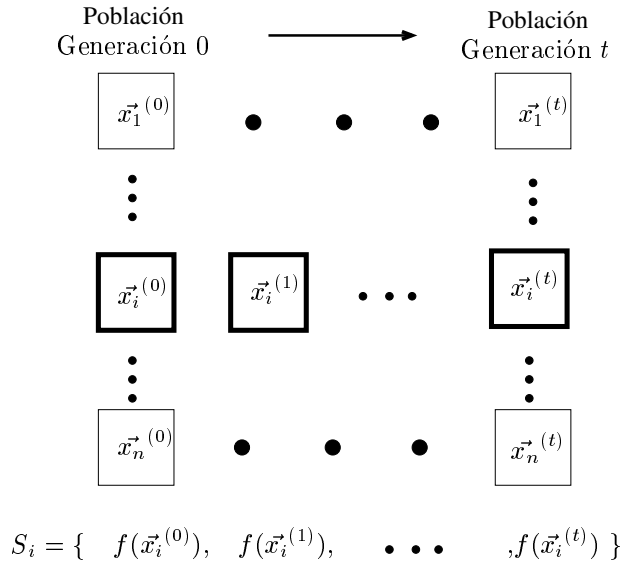


Figura 6.4: Secuencia S_i de valores de aptitud del i-ésimo vector $\vec{x}_i$ en la población durante t generaciones

Por lo tanto, la secuencia S_i guarda la siguiente relación:

$$f(\vec{x}_i^{(t)}) \leq f(\vec{x}_i^{(t-1)}) \leq \ldots \leq f(\vec{x}_i^{(0)}) \tag{6.2}$$

En la secuencia S_i se consideran únicamente problemas de optimización sin restricciones. Ahora consideremos utilizar una función de penalización para manejo de restricciones, tal como se muestra en la ecuación (6.3) en el capítulo 5.

$$\psi(\vec{x}) = \sum_{i=1}^{m} max[0, g_i(\vec{x})]^{\beta} \tag{6.3}$$

La función (6.3) disminuye el grado de penalización, de acuerdo al exponente β, conforme se aproxima a la zona factible; usualmente se emplea un valor de $\beta = 2$, con lo que el grado de penalización aumenta de forma cuadrática conforme se aleja de la zona factible; valores de $\beta > 2$ también son posibles, pero rara vez se emplean. Se puede observar que todas las soluciones factibles no sufren ningún grado de penalización y ésta nunca es negativa. Se consideran además, únicamente problemas de optimización con restricciones de desigualdad, obligando entonces a transformar a las restricciones de igualdad empleando un valor de tolerancia ϵ, tal como se muestra en la ecuación (6.4).

$$|h_j(\vec{x})| - \epsilon \leq 0 \tag{6.4}$$

Empleando a la función de penalización (6.3), se propone una función objetivo extendida $\phi(\vec{x})$ que nos permite transformar un problema restringido a un problema sin restricciones. Esta función se muestra en la ecuación (6.5).

$$\phi(\vec{x}) = f(\vec{x}) + r_g \cdot \psi(\vec{x}) \tag{6.5}$$

La función de penalización mostrada en la ecuación (6.5) es un caso especial de la forma general de las funciones de penalización (ecuación (5.1)), en donde únicamente se emplea un coeficiente de penalización global r_g; el cual, se aplica de igual modo a las violaciones en cada una de las restricciones.

Se observa de la ecuación (6.5), que si el valor de r_g es muy pequeño, una solución infactible puede no ser penalizada lo suficiente. Así se evitará que el proceso de búsqueda encuentre la zona factible. Por otro lado, si r_g es muy grande, es probable que una solución factible sea encontrada rápidamente evitando la correcta exploración de la región infactible. El valor de r_g no sólo es dependiente del problema sino también de la etapa del proceso evolutivo. Por tal motivo, determinar su valor óptimo es difícil en la mayoría de los casos y representa un problema de optimización por sí mismo.

Se observa de la ecuación (6.5) que si $r_g = 0$, no existe penalización alguna por la violación de restricciones y únicamente se considera en la optimización a la función objetivo. Por otro lado, si $r_g < 0$ entonces $\phi(\vec{x})$ favorece con un mejor valor de aptitud a las soluciones infactibles, provocando que el algoritmo converja a zonas no factibles del espacio de búsqueda. Por lo tanto, se establece que el valor de r_g debe ser mayor a cero.

Considerando a la función $\phi(\vec{x})$ se tiene la siguiente secuencia $S_{r_i} = \{\phi(\vec{x}_i^{(1)}), \phi(\vec{x}_i^{(2)}), \ldots, \phi(\vec{x}_i^{(t)})\}$ de valores de aptitud del i-ésimo vector $\vec{x}_i$ en la población durante t generaciones. Esta secuencia guarda la siguiente relación.

$$\phi(\vec{x}_i^{(t)}) \leq \phi(\vec{x}_i^{(t-1)}) \leq \ldots \leq \phi(\vec{x}_i^{(1)}) \tag{6.6}$$

Sustituyendo a la expresión de la función $\phi(\vec{x}_i)$ en la relación anterior tenemos:

$$f(\vec{x}_i^{(t)}) + r_g \cdot \psi(\vec{x}_i^{(t)}) \leq f(\vec{x}_i^{(t-1)}) + r_g \cdot \psi(\vec{x}_i^{(t-1)}) \leq \ldots \leq f(\vec{x}_i^{(1)}) + r_g \cdot \psi(\vec{x}_i^{(1)}) \tag{6.7}$$

Considerando ahora el caso cuando $\vec{x}_i^{(t+1)} = \vec{u}_i^{(t)}$ para la generación $t+1$ y tomando en cuenta la última relación entre $\phi(\vec{u}_i^{(t)})$ y $\phi(\vec{x}_i^{(t)})$, se obtiene:

$$f(\vec{u}_i^{(t)}) + r_g \cdot \psi(\vec{u}_i^{(t)}) \leq f(\vec{x}_i^{(t)}) + r_g \cdot \psi(\vec{x}_i^{(t)}) \tag{6.8}$$

Restando en ambos lados de la desigualdad anterior el término $f(\vec{u}_i^{(t)}) + \psi(\vec{x}_i^{(t)})$ se tiene la siguiente relación:

$$(\psi(\vec{u}_i^{(t)}) - \psi(\vec{x}_i^{(t)})) \cdot r_g \leq f(\vec{x}_i^{(t)}) - f(\vec{u}_i^{(t)}) \tag{6.9}$$

Considerando el caso cuando $\psi(\vec{u}_i^{(t)}) > \psi(\vec{x}_i^{(t)})$ se obtiene.

$$0 < r_g \leq \frac{f(\vec{x}_i^{(t)}) - f(\vec{u}_i^{(t)})}{\psi(\vec{u}_i^{(t)}) - \psi(\vec{x}_i^{(t)})}$$

Y por lo tanto $f(\vec{x}_i^{(t)}) > f(\vec{u}_i^{(t)})$. Está claro que cuando $f(\vec{x}_i^{(t)}) < f(\vec{u}_i^{(t)})$ entonces $\phi(\vec{u}_i^{(t)}) > \phi(\vec{x}_i^{(t)})$ por lo que la solución candidata generada $\vec{u}_i^{(t)}$ no forma parte de la población en la generación $t+1$ y $\vec{x}_i^{(t+1)} = \vec{x}_i^{(t)}$.

Cabe aclarar que se consideran únicamente los casos cuando $\psi(\vec{u}_i^{(t)}) - \psi(\vec{x}_i^{(t)}) \neq 0$ y $f(\vec{x}_i^{(t)}) - f(\vec{u}_i^{(t)}) \neq 0$.

Ahora, tomando en cuenta el caso cuando $\psi(\vec{u}_i^{(t)}) < \psi(\vec{x}_i^{(t)})$ se obtiene.

$$0 < \frac{f(\vec{x}_i^{(t)}) - f(\vec{u}_i^{(t)})}{\psi(\vec{u}_i^{(t)}) - \psi(\vec{x}_i^{(t)})} < r_g$$

Por lo tanto $f(\vec{x}_i^{(t)}) < f(\vec{u}_i^{(t)})$, y queda claro que cuando $f(\vec{x}_i^{(t)}) > f(\vec{u}_i^{(t)})$ entonces $\phi(\vec{u}_i^{(t)}) > \phi(\vec{x}_i^{(t)})$ y $\vec{x}_i^{(t+1)} = \vec{x}_i^{(t)}$.

Estableciendo el valor $\chi = \left| \frac{f(\vec{x}_i^{(t)}) - f(\vec{u}_i^{(t)})}{\psi(\vec{u}_i^{(t)}) - \psi(\vec{x}_i^{(t)})} \right|$ se tiene que $\vec{x}_i^{(t+1)} = \vec{u}_i^{(t)}$ cuando $\psi(\vec{u}_i^{(t)}) > \psi(\vec{x}_i^{(t)})$ y $0 < r_g \leq \chi$. Por otro lado, cuando $\psi(\vec{u}_i^{(t)}) < \psi(\vec{x}_i^{(t)})$ y $0 < \chi < r_g$ también

$\vec{x}_i^{(t+1)} = \vec{u}_i^{(t)}$.

Considerando las demás relaciones entre $f(\vec{x}_i^{(t)}), f(\vec{u}_i^{(t)})$ y $\psi(\vec{u}_i^{(t)}), \psi(\vec{x}_i^{(t)})$ se obtiene la siguiente tabla que describe las condiciones que determinan al vector $\vec{x}_i^{(t+1)}$ en la siguiente generación.

		$0 < r_g \leq \chi$	$0 < \chi < r_g$
$f(\vec{u}_i^{(t)}) < f(\vec{x}_i^{(t)})$	$\psi(\vec{u}_i^{(t)}) < \psi(\vec{x}_i^{(t)})$	$\mathbf{\vec{u}_i}^{(t)}$	$\mathbf{\vec{u}_i}^{(t)}$
$f(\vec{u}_i^{(t)}) < f(\vec{x}_i^{(t)})$	$\psi(\vec{u}_i^{(t)}) > \psi(\vec{x}_i^{(t)})$	$\mathbf{\vec{u}_i}^{(t)}$	$\vec{x}_i^{(t)}$
$f(\vec{u}_i^{(t)}) > f(\vec{x}_i^{(t)})$	$\psi(\vec{u}_i^{(t)}) < \psi(\vec{x}_i^{(t)})$	$\vec{x}_i^{(t)}$	$\mathbf{\vec{u}_i}^{(t)}$
$f(\vec{u}_i^{(t)}) > f(\vec{x}_i^{(t)})$	$\psi(\vec{u}_i^{(t)}) > \psi(\vec{x}_i^{(t)})$	$\vec{x}_i^{(t)}$	$\vec{x}_i^{(t)}$

Tabla 6.1: Condiciones que determinan al vector $\vec{x}_i^{(t+1)}$

De la tabla anterior se observa que $\vec{x}_i^{(t+1)} = \vec{u}_i^{(t)}$ cuando $0 < r_g \leq \chi$ y $f(\vec{u}_i^{(t)}) < f(\vec{x}_i^{(t)})$ independientemente de la relación que guarde $\psi(\vec{u}_i^{(t)})$ y $\psi(\vec{x}_i^{(t)})$. A este tipo de comparación se le denomina **dominada por la función objetivo**, dado que la función f determina la comparación.

Por otro lado, se observa de la tabla 6.1 que también $\vec{x}_i^{(t+1)} = \vec{u}_i^{(t)}$ cuando $0 < \chi < r_g$ y $\psi(\vec{u}_i^{(t)}) < \psi(\vec{x}_i^{(t)})$ independientemente de la relación que guarde $f(\vec{u}_i^{(t)})$ y $f(\vec{x}_i^{(t)})$. Se denomina a este tipo de comparación como **dominada por la función de penalización**, dado que la función ψ determina el sentido de la desigualdad (6.8).

Estos son los únicos casos donde $\vec{x}_i^{(t+1)} = \vec{u}_i^{(t)}$. Por ende, las desigualdades únicamente están dominadas por la función objetivo o por la función de penalización dependiendo si el valor $0 < r_g < \chi$ o $r_g > \chi$. Es decir, para poder determinar el sentido de la desigualdad, bastaría comparar únicamente los valores de la función objetivo o el grado de violación de restricciones dependiendo del valor de r_g.

Existen t valores de χ_j para la secuencia S_{r_i} de valores de aptitud del vector $\vec{x}_i$. Si el valor de $r_g > max[\chi_1, \ldots, \chi_t]$, entonces la relación que guarda la secuencia S_{r_i} está basada únicamente en los valores de la función de penalización; este caso se denomina **sobre-penalización**. Por otro lado, si el valor de $r_g < min[\chi_1, \ldots, \chi_t]$, entonces la relación que guarda la secuencia S_{r_i} está determinada exclusivamente por la función objetivo; a este caso se nombra **sub-penalización**. Se ha establecido experimentalmente que la utilidad de las técnicas basadas en **sub-penalización** o **sobre-penalización** es muy limitada; por lo tanto, una técnica competitiva de manejo de restricciones debe evitar cualquiera de los dos

casos.

Determinar el valor de r_g que induzca un balance adecuado, y así evitar a la sobre- o sub-penalización es difícil. Esto debido a que no es posible conocer *a priori* los valores de χ_j. Así, inducir un balance entre la función objetivo y la función de penalización a través del valor de r_g es complejo.

Es posible obtener un balance equivalente que el emplear una función objetivo extendida $\phi(\vec{x})$ (ecuación (6.5)) y adaptando el parámetro r_g , si las comparaciones de los individuos $\vec{u}_i^{(t)}, \vec{x}_i^{(t)}$ están basadas únicamente en los valores de $\psi(\vec{u}_i^{(t)}), \psi(\vec{x}_i^{(t)})$ o $f(\vec{u}_i^{(t)}), f(\vec{x}_i^{(t)})$. De tal manera, es posible controlar la proporción de comparaciones basadas en la función objetivo o en la función de penalización empleando un ensayo aleatorio independiente de dos resultados (éxito o fracaso) con cierta probabilidad de éxito P_f, donde un éxito corresponde a emplear a la función objetivo sin importar el grado de violación de restricciones; y un fracaso (con probabilidad P_p) implicaría utilizar a la función de penalización no importando los valores de la función objetivo; en tal caso $P_p = 1 - P_f$.

Dado un valor constante de P_f, la probabilidad $b(k; t, P_f)$ de que a lo largo de t generaciones se logren exactamente k comparaciones basadas en la función objetivo está determinada por:

$$b(k; t, P_f) = \binom{t}{k} P_f^k (1 - P_f)^{t-k}$$

En particular, la probabilidad de tener una sobre-penalización o una sub-penalización es $(1 - P_f)^t$ y P_f^t respectivamente. Ambas cantidades decrecen exponencialmente conforme el número de generaciones t aumenta. Por ejemplo, para $P_f = 0.45$ después de 100 generaciones, la probabilidad de una sub-penalización es aproximadamente 4.5×10^{-18} mientras que la probabilidad de una sobre-penalización es cercana a 1.0×10^{-13}.

Cabe mencionar que se emplea este tipo de ensayo debido a que no se sabe de antemano el orden de aplicación del criterio de comparación. Se espera que la media del número de comparaciones basadas en la función objetivo sea $\mu_f = t \times P_f$; es decir, directamente proporcional a P_f cuando P_f es constante.

A continuación, se muestra el mecanismo de selección propuesto entre los individuos $\vec{x}_i^{(t)}$ y $\vec{u}_i^{(t)}$ donde $flip(P_f)$ es un evento aleatorio independiente de dos resultados (verdadero o falso) con una probabilidad P_f.

```
1 x_i^(t+1) = x_i^(t) ;
2 if ψ(u_i^(t)) = ψ(x_i^(t)) = 0 ∨ flip(P_f) then
3 |   if f(u_i^(t)) < f(x_i^(t)) then
4 |   |   x_i^(t+1) = u_i^(t) ;
5 else
6 |   if ψ(u_i^(t)) < ψ(x_i^(t)) then
7 |   |   x_i^(t+1) = u_i^(t) ;
```

Algoritmo 11: Mecanismo de selección propuesto para manejo de restricciones en la ED

Empleando el criterio de selección anterior, no sólo se logra un balance entre la función objetivo y la función de penalización, sino que además mantiene un balance entre individuos factibles e infactibles dentro de toda la población en cada generación; aún cuando el valor de P_f no sea constante a lo largo de todas las generaciones.

Jimenez et al. [44] y Deb et al. [19] propusieron de manera independiente el siguiente criterio de comparación empleado en una selección de tipo torneo el cual ha sido ampliamente usado en los algoritmos evolutivos para manejo de restricciones.

1. Una solución factible es preferible que cualquier solución infactible.
2. Entre dos soluciones factibles, aquella con mejor valor de la función de objetivo es preferible.
3. Si dos soluciones son infactibles, es preferible a aquella que posea una menor violación de restricciones.

Este criterio de comparación es un caso especial del mecanismo de selección propuesto (algoritmo 11) donde $P_f = 0$; con este valor de P_f, una selección tipo torneo genera únicamente **sobre-penalizaciones**. El mecanismo propuesto no es un criterio de comparación exclusivo de la Evolución Diferencial, sino que puede emplearse como un criterio de comparación en otros algoritmos evolutivos. Ésta no es la primer propuesta donde se busca un balance entre soluciones factibles e infactibles; las JE introducen directamente un ordenamiento deseado en los individuos a través de un procedimiento similar al ordenamiento de burbuja; la comparación entre individuos adyacentes se basa tanto en la función objetivo como en la función de penalización y tal elección se efectúa de manera aleatoria [80]. En la estrategia evolutiva de un solo miembro (SMES) propuesta por Mezura et al. [60], se emplea un mecanismo que mantiene en la población a un número dado de las mejores soluciones infactibles encontradas en el proceso de búsqueda. Finalmente, en la propuesta de Lampinen

et al. [55] no existe un balance entre soluciones factibles e infactibles.

En lugar de mantener un valor constante de P_f durante todo el proceso de búsqueda, éste es disminuido de manera lineal a lo largo de G_{max} generaciones. Al final, lo que buscamos son las soluciones factibles y no las infactibles; así, el balance de la población final no es de importancia. Por tal motivo, este valor decrece linealmente conforme el proceso de búsqueda avanza, favoreciendo en su inicio a la exploración del espacio de búsqueda y conforme pasan las generaciones se favorece a la explotación del mismo. Dada la característica de la Evolución Diferencial de encontrar rápidamente zonas de buena aptitud, no es necesario mantener un valor de P_f cercano a 0.5 durante todo el proceso de búsqueda (como en las Jerarquías Estocásticas donde $P_f = 0.45$), el cual puede evitar la convergencia de la ED. Así tampoco es necesario mantener un valor P_f muy pequeño (como en la SMES donde $P_f = 0.03$), ya que puede evitar la correcta exploración del espacio de búsqueda y generar convergencia prematura. De tal manera, el valor de $P_f^{(t)}$ en la generación t se calcula de la siguiente manera.

$$P_f^{(t+1)} = P_f^{(t)} - \Delta_{P_f} \tag{6.10}$$

Donde $\Delta_{P_f} = \left(\frac{P_f^{(0)} - P_f^{(G_{max})}}{G_{max}}\right)$. De manera experimental se observó que un valor de $P_f^{(0)} = 0.55$ y $P_f^{(G_{max})} = 0.03$ proveen el balance necesario para resolver el conjunto de trece problemas de prueba en optimización con restricciones. Cabe mencionar que esto es muy diferente a otros intentos de modificar los factores de penalización de manera dinámica (penalizaciones dinámicas); por ejemplo, el método "Adaptive Segregational Constraint Handling" propuesto por Sana Ben Hamida y Marc Schoenauer[37], ya que el valor de P_f no es dependiente del problema.

Dado que el valor de P_f no es constante, entonces las probabilidad de tener una sobrepenalización o una sub-penalización ya no son $(1 - P_f)^{G_{max}}$ y $P_f^{G_{max}}$ respectivamente. Empleando la regla del producto, se calcula la probabilidad (P_{sub}) de una sub-penalización.

$$\begin{aligned} P_{sub} &= P_f^{(0)} \cdot P_f^{(1)} \cdots P_f^{(G_{max})} && (6.11)\\ &= P_f^{(0)} \cdot (P_f^{(0)} - \Delta_{P_f}) \cdots (P_f^{(0)} - G_{max} \cdot \Delta_{P_f}) && (6.12)\\ &= \prod_{i=0}^{G_{max}} (P_f^{(0)} - i \cdot \Delta_{P_f}) && (6.13) \end{aligned}$$

De manera análoga se calcula la probabilidad (P_{sobre}) de una sobre-penalización.

$$P_{sobre} = (1 - P_f^{(0)}) \cdot (1 - P_f^{(1)}) \cdots (1 - P_f^{(G_{max})}) \tag{6.14}$$

$$= (1 - P_f^{(0)}) \cdot (1 - (P_f^{(0)} - \Delta_{P_f})) \cdots (1 - (P_f^{(0)} - G_{max} \cdot \Delta_{P_f})) \tag{6.15}$$

$$= \prod_{i=0}^{G_{max}} (1 + i \cdot \Delta_{P_f} - P_f^{(0)}) \tag{6.16}$$

Ambas cantidades decrecen conforme $G_{max} \to \infty$. En la tabla 6.2 se muestra una comparativa de P_{sub} y P_{sobre} conforme G_{max} crece; se observa que para $G_{max} = 100$ los dos eventos son muy poco probables aunque la diferencia entre ambos es de varios órdenes de magnitud. Esto es por el sesgo introducido conforme avanza el proceso evolutivo, ya que al final se busca llegar a la zona factible.

$G_{max} =$	10	55	100
P_{sub}	$6.9e^{-11}$	$8.7e^{-36}$	$9.9e^{-64}$
P_{sobre}	$1.6e^{-2}$	$1.2e^{-9}$	$8.5e^{-17}$

Tabla 6.2: Probabilidades de sobre- y sub-penalización del mecanismo de manejo de restricciones propuesto

6.3. Algoritmo Propuesto

En la figura 6.5 se muestra un diagrama que ilustra el proceso evolutivo de un vector $\vec{x}_i$ de la población en una sola generación.

Es importante mencionar que $\vec{x}_{best}^{(g)}$ es el vector que posee el mejor valor de aptitud dentro de la generación g; es decir, no es la mejor solución encontrada a lo largo de todo el proceso de búsqueda. Esto difiere de otras propuestas que emplean *elitismo*. La Evolución Diferencial emplea un esquema de mutación que incorpora información global de la población. De hecho, la velocidad de propagación de dicha información caracteriza a la ED; por tal motivo, emplear un esquema de *elitismo* tradicional provoca en la mayoría de los casos convergencia prematura.

Un aspecto importante de la nueva propuesta, es que el mecanismo de selección no es determinista, a diferencia de la versión original de la ED. De hecho, la forma de mantener el *elitismo* es empleando ese mecanismo de selección determinista.

Es posible que al emplear el esquema de manejo de restricciones propuesto, la mejor solución en la generación g se pierda y no forme parte de la generación $g+1$, aún cuando corresponda al valor óptimo global. Es importante mencionar que el algoritmo debe ser capaz de *entrar* y *salir* de la zona factible, esto con el fin de explorar la frontera. Como se mencionó en el capítulo 3, en muchos problemas restringidos es posible que el óptimo se localice en el borde de la zona factible. Esto difiere de la propuesta de Lampinen [55], donde el criterio de optimalidad empleado no permite movimientos en la frontera.

Cabe mencionar, que el papel que juega el buscador local es doble: por una parte ayuda a aumentar la capacidad de exploración del algoritmo y por otro lado disminuye la pérdida de información obtenida durante el proceso de búsqueda a consecuencia del proceso de selección estocástico. De hecho, se sugiere emplear un valor de $\lambda \in [2, 7]$. Este valor se fijó a $\lambda = 5$ para los problemas de prueba empleados. Un valor mayor a cinco no mejora el rendimiento sin embargo, sí incrementa por un factor el número de evaluaciones de la función objetivo; un valor menor a cinco disminuye la calidad de los resultados de manera sustancial y conforme $\lambda \to 1$ el algoritmo no converge a una solución única ni factible.

Este rango de valores concuerda con un estudio realizado por Deb et. al [21] donde se comparan diferentes esquemas de recombinación empleado un algoritmo evolutivo de estado uniforme. En este estudio, una solución genera más de un descendiente. Los resultados experimentales mostraron que no es necesario generar muchos descendientes, pues se obtienen los mejores resultados cuando $2 \leq \lambda \leq 10$ aún cuando se emplearon en el estudio tres diferentes tipos de cruza.

Finalmente, el algoritmo 12 muestra en su forma completa a la propuesta de Evolución Diferencial. Es importante mencionar que de los parámetros de control el usuario define, únicamente: el tamaño de la población μ, el número máximo de generaciones G_{max}, el número de descendientes λ por individuo y el parámetro de control de la cruza discreta CR. Los demás parámetros se proponen de la siguiente forma: $F_\alpha = 0.8$, $F_\beta = 0.1$, $P_f^{(0)} = 0.55$, $P_f^{(G_{max})} = 0.03$ y éstos no cambian en todos los problemas de prueba tanto con restricciones como sin ellas.

A diferencia de la propuesta de Rainer y Storn [75] donde se sugiere emplear poblaciones de tamaño veinte veces más grande que el número de variables del problema $(10 \times n)$ o más, en nuestra propuesta no establecemos una relación entre μ y n. De hecho, como se verá el capítulo siguiente, es posible resolver problemas con miles de variables de decisión aún empleando una población $\mu = 15$ individuos. De manera experimental se observó que poblaciones con tamaño $15 \leq \mu \leq 40$ son suficientes para la mayoría de los problemas de prueba empleados (con y sin restricciones). Esto concuerda con la idea de la búsqueda

dispersa [31] donde se mantiene un conjunto reducido de soluciones élite así como en la propuesta de Deb [21], que emplea algoritmos de estado uniforme con poblaciones pequeñas de individuos élite.

Sin embargo, si existe una relación entre el número de variables del problema y el número máximo de generaciones que emplea el algoritmo. Esto se detallará en el próximo capítulo.

6.4. Resumen del Capítulo

La propuesta presentada de la Evolución Diferencial añade al algoritmo original estrategias de diseño que extienden sus características iniciales. Dichas características son las siguientes:

- Se incorpora como estrategia de diseño emplear la información de la zona de mejor aptitud encontrada como guía del proceso de búsqueda.
- Se incorpora información de la trayectoria de una solución dentro de la población en la generación de sus descendientes.
- Se aumenta la capacidad de explotación del algoritmo empleando un buscador local.
- Se emplea un mecanismo de manejo de restricciones que únicamente modifica el proceso de selección. El resto de la estructura del algoritmo empleado en optimización global sin restricciones no es modificada.
- El mecanismo de manejo de restricciones evita la **sobre-** y **sub-** penalización.

Las estrategias de diseño son una forma *simple* de incorporar información del dominio; de otra manera se deja al algoritmo que por sí mismo las descubra, esto último a costa de una reducción en el rendimiento.

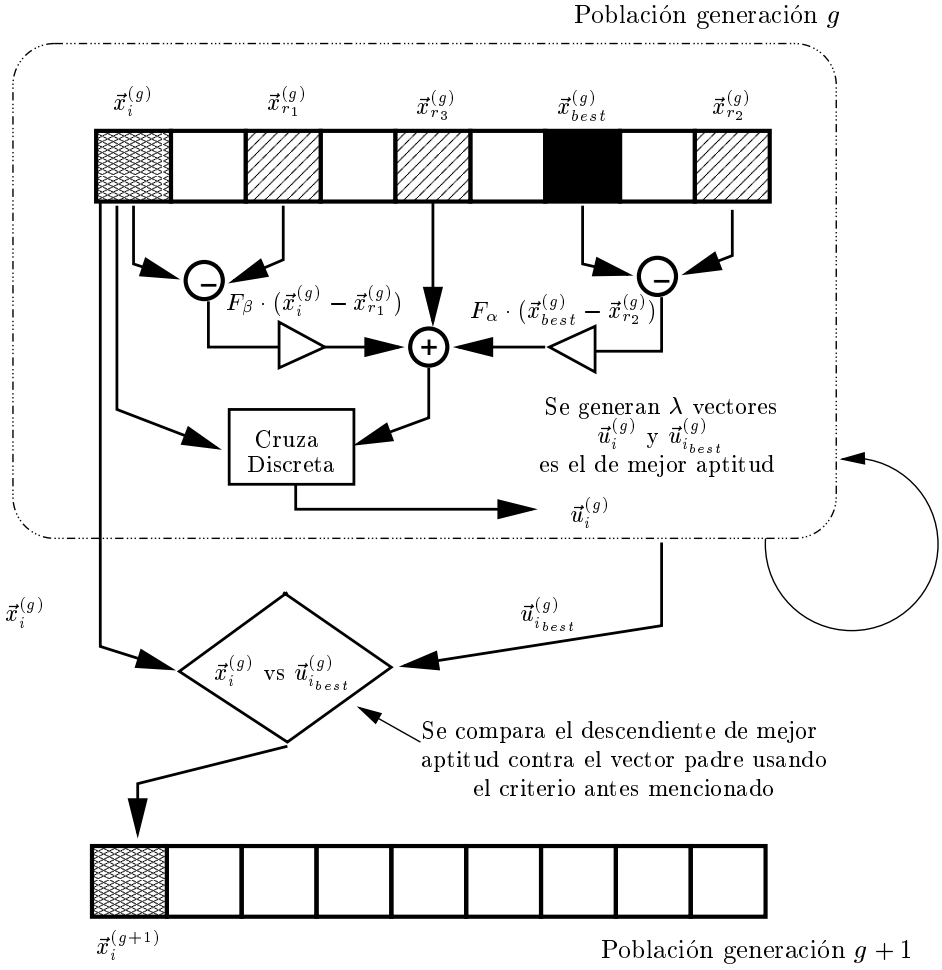


Figura 6.5: Ilustración del proceso evolutivo del $\vec{x}_i^{(g)}$ en una sola generación, para el algoritmo propuesto de Evolución Diferencial

Input: μ, λ, G_{max}, CR, F_α, F_β, $P_f^{(0)}$, $P_f^{(G_{max})}$
Result: $\vec{x}_{best}^{(G_{max})}$

```
1  Inicializar(P_g ← {x⃗_1^(0), ..., x⃗_μ^(0)})
2  Δ_Pf = ((P_f^(0) − P_f^(G_max)) / G_max)
3  for g ← {1, ..., G_max} do
4      for i ← {1, ..., μ} do
5          for k ← {1, ..., λ} do
6              r_1, r_2, r_3 ∈ {1, ..., μ} aleatoriamente seleccionados,
7                                   donde r_1 ≠ r_2 ≠ r_3 ≠ i ;
8              j_rand ∈ {1, ..., n} aleatoriamente seleccionado ;
9              for j ← {1, ..., n} do
10                 if U_j[0,1] < CR or j = j_rand then
11                     u_{i,j}^(g) ← x_{r3,j}^(g) + F_α · (x_{best,j}^(g) − x_{r2,j}^(g)) + F_β · (x_{i,j}^(g) − x_{r1,j}^(g))
12                 else
13                     u_{i,j}^(g) ← x_{i,j}^(g)
14             if k = 1 ∨ f(u⃗_i^(g)) ≤ f(u⃗_{i_best}^(g)) then
15                 u⃗_{i_best}^(g) = u⃗_i^(g)
16         x⃗_i^(g+1) = x⃗_i^(g)
17         if ψ(x⃗_i^(g)) = ψ(u⃗_{i_best}^(g)) = 0 ∨ flip(P_f^(g)) then
18             if f(u⃗_{i_best}^(g)) ≤ f(x⃗_i^(g)) then
19                 x⃗_i^(g+1) = u⃗_{i_best}^(g)
20         else
21             if ψ(u⃗_{i_best}^(g)) ≤ ψ(x⃗_i^(g)) then
22                 x⃗_i^(g+1) = u⃗_{i_best}^(g)
23     P_f^(g+1) = P_f^(g) − Δ_Pf
```

Algoritmo 12: Algoritmo de evolución diferencial propuesto para optimización global con restricciones (**newde**) *(suponiendo un problema de minimización)*

Capítulo 7

Resultados Experimentales

En la primera parte del capítulo se muestran los resultados de un estudio comparativo del nuevo modelo de Evolución Diferencial **newde** contra los modelos: **rand/1/bin** y **best/1/bin**, empleando en ello a un conjunto de funciones de prueba estándar utilizadas en la optimización global sin restricciones.

En la segunda parte se muestra una comparativa de la propuesta de Evolución Diferencial contra los algoritmos más representativos del área de optimización en espacios restringidos en la computación evolutiva.

7.1. Nuevo Modelo de Evolución Diferencial

Con la finalidad de evaluar el desempeño del nuevo modelo de Evolución Diferencial **(newde)**, se empleó el conjunto de funciones de prueba estándar en la literatura de optimización global sin restricciones [92]. Este conjunto consta de veintitrés funciones de prueba de las cuales, trece pueden ser escaladas a un arbitrario número de variables de decisión. A excepción de la función de prueba $f08$, en todas las demás la localización del valor óptimo no depende de la dimensionalidad del problema. La descripción de las funciones se encuentra en el Apéndice A al final de este documento.

El nuevo modelo se comparó contra el modelo **rand/1/bin**, debido a que es el modelo original de la Evolución Diferencial, el más ampliamente usado y uno de los que presenta mejores resultados. Además, también se comparó contra el modelo **best/1/bin** que mostró el mejor desempeño (calidad de soluciones y número de evaluaciones de la función objetivo) al compararse contra otros modelos de ED (véase capítulo 4).

7.1.1. Parámetros de Control Empleados

La ED utiliza un parámetro de control F, el cual regula las magnitudes relativas de las diferencias del vector mutación. En la literatura, su valor se propone en el intervalo $[0, 1)$ [75]. Pero, de manera experimental se determinó que el comportamiento del algoritmo no sufre efectos significativos adversos al generarlo de manera aleatoria en el intervalo $[0.3, 0.9]$ durante cada generación [65]. De tal suerte, en esta serie de experimentos se empleó esta propuesta con los modelos **rand/1/bin**, **best/1/bin**, haciéndose innecesario establecer un valor predeterminado para este parámetro. Para el caso del nuevo modelo, no existe un único parámetro F, sino que se existen dos parámetros F_α, F_β que regulan la dirección de búsqueda en proporción de la mejor solución en la población o de la mejor solución en la trayectoria de la solución padre. De manera experimental se determinó que los valores $F_\alpha = 0.8$ y $F_\beta = 0.1$ son suficientes para la solución de este conjunto de funciones de prueba.

Se determinó también por cada par **modelo-función** el tamaño mínimo requerido de la población para resolver cada problema dado un número máximo de evaluaciones. El tamaño de la población usado por los modelos **rand/1/bin** y **best/1/bin** en todas las funciones de prueba es $\mu = 60$; el cual se determinó como el máximo de los tamaños mínimos de población de cada función. Para el conjunto de funciones de prueba empleadas, el modelo **newde** requiere poblaciones de individuos no mayores a 40. De hecho el tamaño de población empleado es de $\mu = 15$ individuos, pues poblaciones grandes son innecesarias. Como se mencionó en el capítulo anterior, esto contradice lo propuesto por Storn and Rice [75].

El parámetro adicional $\lambda \in [2, 7]$ del modelo **newde**, determina el número de individuos a generar por cada solución dentro de la población. El valor empleado fue $\lambda = 2$, y se determinó tras realizar pruebas exhaustivas para cada función de prueba. El número máximo de evaluaciones de la función objetivo se fijó a $120,000$. Así, el número de generaciones (G_{max}) se ajustó con la finalidad de emplear el mismo número de evaluaciones en todos los modelos, quedando los valores de G_{max} de la siguiente forma:

- **rand/1/bin:** $G_{max} = 2000$
- **best/1/bin:** $G_{max} = 2000$
- **newde:** $G_{max} = 4000$

En cada uno los modelos empleados se utiliza una recombinación discreta binomial; el parámetro CR se define dentro del intervalo $[0, 1]$ dado que representa una probabilidad. En la mayoría de los casos, el comportamiento del algoritmo se muestra muy susceptible al valor de éste. Por lo tanto, se empleó una búsqueda exhaustiva para determinar el mejor valor de CR para cada par **modelo-función**.

modelo	$f01$	$f02$	$f03$	$f04$	$f05$	$f06$	$f07$	$f08$	$f09$	$f10$	$f11$	$f12$	$f13$
rand/1/bin	0.9	0.2	0.9	0.8	0.9	0.8	0.9	0.0	0.0	0.9	0.9	0.0	0.1
best/1/bin	0.7	0.7	0.8	0.3	0.3	0.2	0.7	0.0	0.0	0.3	0.1	0.1	0.2
newde	0.0	0.0	1.0	0.2	0.9	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.0

Tabla 7.1: Valores del parámetro CR empleados para cada función de prueba, los cuales fueron determinados de manera exhaustiva

En la tabla 7.1 se muestran los valores de CR empleados para cada par **modelo-función**. Es importante notar que en todos los casos $CR \in [0.7, 1.0]$ o $CR \in [0.0, 0.3]$; de hecho, emplear valores intermedios degrada el desempeño del algoritmo. En cualquier caso, esto indica que únicamente unos pocos valores de las variables de decisión sufren cambio; la mayoría de la información es transferida del padre o del vector mutación directamente al descendiente.

A continuación se listan los valores de los principales parámetros de control (tamaño de población (μ), número máximo de generaciones G_{max} y número de descendientes λ) empleados en este estudio comparativo.

modelo	μ	G_{max}	λ
rand/1/bin	60	2000	no aplica
best/1/bin	60	2000	no aplica
newde	15	4000	2

7.1.2. Diseño Experimental

En cada ejecución del algoritmo de Evolución Diferencial se genera una solución candidata, la cual representa al mejor individuo encontrado en el proceso de búsqueda. Se espera que este valor corresponda con el valor óptimo global del problema en cuestión.

Así, el experimento consistió en efectuar 100 ejecuciones independientes (con diferente semilla inicial en la generación de números aleatorios) para cada par **modelo-problema**, empleando los valores de parámetros antes mencionados.

En las figuras 7.1 y 7.2 se muestran el histograma en frecuencias y la gráfica cuantil normal del conjunto de soluciones de 100 ejecuciones independientes para el modelo **rand/1/bin** en el problema $f09$.

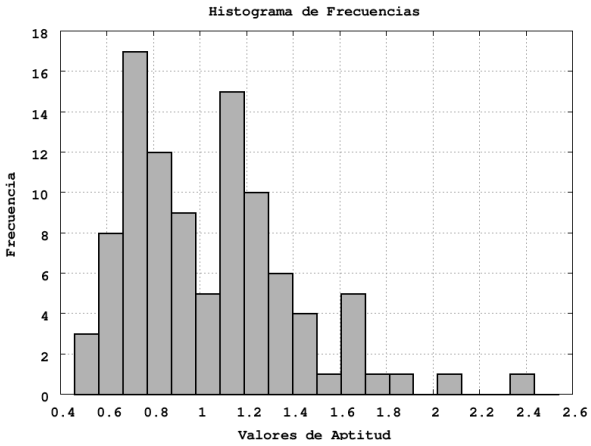


Figura 7.1: Histograma de frecuencias

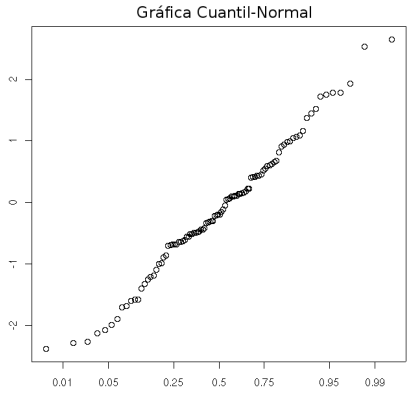


Figura 7.2: Gráfica Cuantil Normal

Se observa de las gráficas que la muestra compuesta de 100 soluciones resultado de ejecuciones independientes, no se ajusta a una forma normal. Por lo tanto, medidas estadísticas como la media o la desviación estándar no son significativas en esta muestra.

Se observa un comportamiento muy similar en los demás conjuntos solución por cada uno de lo pares **modelo-problema**. Por tal motivo, es necesario emplear otros métodos que cuantifiquen el comportamiento del algoritmo.

7.1.3. Métodos Estadísticos de Cómputo Intensivo

El análisis de datos tradicional (derivar la distribución muestral y calcular con ella la probabilidad de la muestra) se fundamenta en diferentes suposiciones, por ejemplo: los datos son independientes, los datos se ajustan a una forma de distribución específica, un estimador posee una distribución dada, etc. En algunas ocasiones estas suposiciones están justificadas, ya sea por conocimiento previo o por verificación empírica. Pero, en los casos donde tales suposiciones no se cumplen, su relevancia es cuestionable.

Existe una clase de métodos basados en la simulación del proceso de muestreo, los cuales son llamados *métodos estadísticos de cómputo intensivo* dentro de los cuales se encuentran las pruebas de permutación, métodos de Monte Carlo, métodos de Bootstrap, etc. En general, las suposiciones en las que se basan estos métodos son menores que los métodos tradicionales.

A continuación se listan algunas de las ventajas de estos nuevos métodos [66]:

- Menores suposiciones: no se requiere que las muestras se ajusten a una distribución normal o que las muestras sean de gran tamaño.
- Mayor precisión: las pruebas de permutación y algunos métodos de bootstrap son más precisos en la práctica que los métodos clásicos.
- Generalidad: los métodos de remuestreo son muy similares para un amplio rango de estadísticas, y no requieren una fórmula por cada una de ellas.

7.1.4. Método de Bootstrap

Los métodos de bootstap son procedimientos de simulación del proceso de muestreo que han sido ampliamente usados en la determinación de estimadores estadísticos, en la construcción de intervalos de confianza, en el cómputo de probabilidades en una prueba de hipótesis, etc.

La idea de los métodos de bootstrap consiste en generar una distribución muestral para una estadística dada, empleando un remuestreo de Monte Carlo y considerando a la distribución de la muestra como la distribución de la población misma. La distribución resultante es llamada **Distribución Muestral de Bootstrap** (DMB) y puede ser calculada para casi cualquier estadística; el único requerimiento es que la muestra sea representativa de la población [13]. La validez del bootstrap se basa en que la distribución muestral es el mejor estimador de la distribución de la población. Si una muestra contiene N puntos muestrales, entonces el estimador de máxima probabilidad de la distribución de la población es determinado asignando a cada punto muestral una probabilidad de $\frac{1}{N}$ [13].

1 **for** $i \leftarrow \{1, \ldots, K\}$ **do**
2 Generar un re-muestra S_i^* de la muestra original S con reemplazo, igual probabilidad y mismo tamaño.
3 Calcular y guardar el valor Θ_i^* de la estadística Θ^* de S_i^*.
4 Los valores $\Theta_1^*, \ldots, \Theta_K^*$ componen a la **distribución muestral de Bootstap**.

Algoritmo 13: Esquema general de un procedimiento de bootstrap

En el algoritmo 13 se muestra el esquema general de un procedimiento de bootstrap de K remuestreos, empleando una muestra S de tamaño n con la cual, se busca una DMB para un estimador Θ^* del parámetro de la población Θ. El poder del procedimiento de bootstrap consiste en que se pueden determinar distribuciones para muchas estadísticas, por ejemplo:

la media, la media recortada, la distancia inter-cuartil, etc., esto, sin requerir normalidad en las muestras. Típicamente se emplean 1000 remuestreos en tales procedimientos.

7.1.5. Resultados Experimentales

La media muestral $\overline{x}$ es un estimador no sesgado de la media de la población μ. Así, calculando la media muestral $\overline{x}$ de la distribución muestral de Bootstrap (DMB) usando como estadística a la misma media muestral $\overline{x}$, se obtiene un estimador $\hat{\mu}$ de la media de la población μ, sin asumir condiciones de normalidad en la muestra.

En la mayoría de los casos, la DMB *imita* la forma y extensión de la distribución muestral, pero ésta es centrada en el valor original del parámetro estimado [66]; en este caso, la media de la población. Esta es la forma en que el Bootstrap estima los parámetros de la población original a partir de la distribución muestral. Por lo tanto, la media muestral de la DBM ($\hat{\mu}$) nos da un estimado del comportamiento promedio real del algoritmo para cada función de prueba.

Prob.	***rand/1/bin***	***best/1/bin***	***newde***
f01	***0.0****	***0.0****	***0.0****
f02	***0.0****	***0.0****	***0.0****
f04	*1.386*	*0.0014*	*0.00008**
f06	***0.0****	***0.0****	***0.0****
f07	***0.0****	***0.0****	***0.0****
f08	***0.0****	***0.0****	***0.0****
f09	***0.0****	***0.0****	*0.242551*
f03	*0.03005*	***0.0****	***0.0****
f10	***0.0****	*0.0235*	***0.0****
f05	*17.058*	*37.315*	*4.538**
f11	*0.0022*	*0.0009*	*0.00016**
f12	***0.0****	***0.0****	***0.0****
f13	***0.0****	***0.0****	***0.0****

Tabla 7.2: Estimaciones de la media de aptitud $\hat{\mu}$ para cada uno de los tres modelos de ED en los 13 problemas de prueba.

En la tabla 7.2 se muestran los resultados de las estimaciones de la aptitud media $\hat{\mu}$ de los tres modelos de ED para cada uno de los trece problemas de prueba. En la tabla se agrupan las funciones de acuerdo a sus características de modalidad y separabilidad, véase capítulo 4 ($f01, f02, f04, f06, f07$ unimodales y separables, $f08, f09$ multimodales y separables,

$f03$ unimodal y no separable, $f10, f05, f11, f12, f13$ mutimodales y no separables).

Valores marcados con **negritas** indican que el modelo encuentra al valor óptimo de manera consistente. Valores señalados con * indican los mejores valores de aptitud media encontrados para una función dada. De la tabla 7.2 se observa que en todos los casos, a excepción de $f09$, el modelo **newde** muestra los mejores resultados. Aún cuando el número de evaluaciones de la función objetivo son insuficientes para resolver la función $f05$ (multimodal no separable), el modelo **newde** muestra un mejor comportamiento promedio que las otras dos propuestas.

En las figuras 7.3 y 7.4 se muestran la gráfica en tres dimensiones y la gráfica de curvas de nivel de la instancia de dos variables de decisión para el problema generalizado de Rastrigin $f09$. Esta función es un ejemplo típico de una función no lineal multimodal, cuya dificultad radica en la alta cantidad de óptimos locales (como se aprecia en la figura 7.4). Sin embargo, los modelos de Evolución Diferencial **[best—rand]/1/bin** logran resolverla de manera consistente. El modelo **newde** no logra resolverla consistentemente. En el 78 % de los casos se encuentra el óptimo global, mientras que en el 22 %, el algoritmo queda atrapado en un óptimo local.

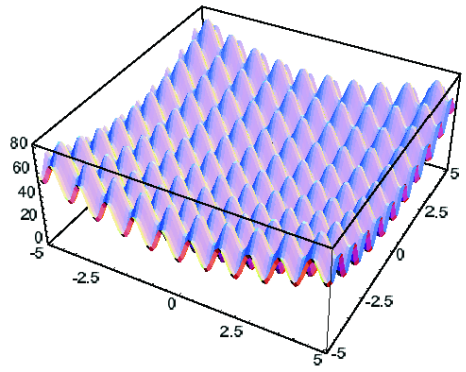


Figura 7.3: Gráfica de la función generalizada de Rastrigin $f09$ (2 variables de decisión)

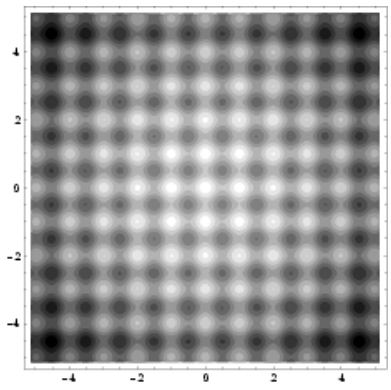


Figura 7.4: Curvas de nivel de la función generalizada de Rastrigin $f09$ (2 variables de decisión)

De la tabla 7.2 se observa que ninguno de los modelos logra resolver a la función generalizada de Rosenbrock $f05$ con $120,000$ evaluaciones. En las figuras 7.5 y 7.6 se muestran la gráfica en tres dimensiones y la gráfica de curvas de nivel de la instancia de dos variables del problema. En la figura 7.5 se observa un comportamiento *suave* de la función; sin embargo, observando la gráfica 7.6 se aprecia que el óptimo global se localiza en un *valle* muy reducido en comparación al rango empleado en las variables de decisión $x_1, x_2 \in [-30, 30]$, haciendo difícil su localización. Sin embargo, en esta función el modelo **newde** muestra un

comportamiento sensiblemente mejor que sus competidores.

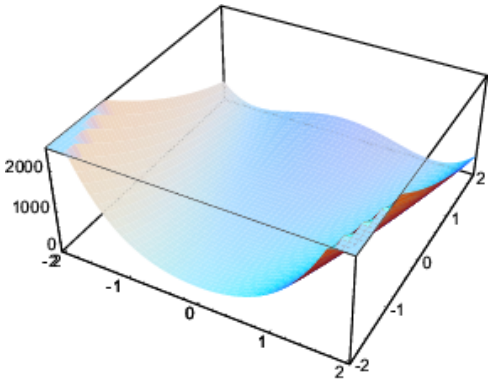


Figura 7.5: Gráfica de la función generalizada de Rosenbrock $f09$ (2 variables de decisión)

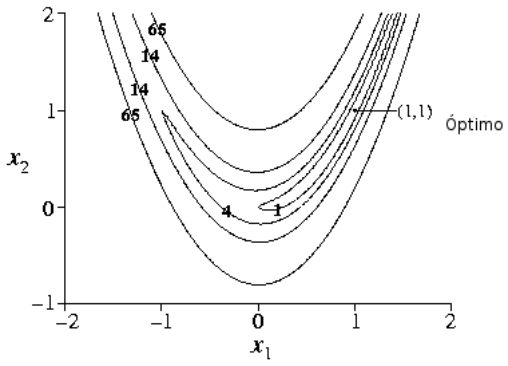


Figura 7.6: Curvas de nivel de la función generalizada de Rosenbrock $f09$ (2 variables de decisión)

7.1.6. Intervalos de Confianza

En la estimación de los parámetros de una población desconocida a partir de una muestra, es deseable poder caracterizar la precisión de las estimaciones de los parámetros. Los intervalos de confianza (IC) proveen una herramienta para esta situación.

Supóngase que no se conoce la media de la población μ, pero para una muestra dada se calcula $\overline{x}$, el cual es un estimador de μ. El valor de μ puede ser menor o mayor que $\overline{x}$, esto es, $\mu = \overline{x} \pm \epsilon$; cuando ϵ es pequeña, se dice que $\overline{x}$ es un buen estimador de μ. Así, para un valor de ϵ, un intervalo de confianza $\overline{x} \pm \epsilon$ con una seguridad del 95 % implica que el valor de μ estará contenido el 95 % de las veces dentro del intervalo en muestreos independientes de la población. Esto es, un intervalo de confianza implica con un grado de seguridad que en el intervalo alrededor de $\overline{x} \pm \epsilon$ estará contenido el valor de μ [13]. Cuando las muestras se ajustan a una distribución normal, es posible emplear la prueba **Z** para calcular los intervalos de confianza para diferentes grados de seguridad empleando únicamente a $\overline{x}$ y S, es decir la media y desviación estándar muestral. De hecho, los IC son simétricos con respecto a $\overline{x}$ porque se considera que la muestra se ajusta a una forma normal.

Cuando las muestras no son normales, el procedimiento de Bootstrap nos permite calcular los intervalos de confianza de diversas formas. Es posible que la muestra no sea normal y sin embargo la distribución muestral de bootstrap sí lo sea. En ese caso, es posible aplicar una prueba **Z** o **t** sobre la DMB directamente. Desafortunadamente esto no siempre se cumple. De hecho, para muchos de los resultados de cada par **modelo-problema** la DMB no se

ajusta a una forma normal. Por tal motivo, otra forma de calcular los intervalos de confianza (IC) es empleando los valores percentiles sobre la DMB. Esto es, si se quiere calcular un IC con 95 % de confianza, se toma como valor mínimo al percentil 2.5 y como valor máximo al percentil 97.5 en la DMB; con esto es posible determinar IC no centrados en la media.

En la figura 7.7 se ilustra el cálculo de los IC. Cuando la DMB se ajusta a una forma normal, los intervalos de confianza empleando percentiles, la prueba **Z** o la prueba **t** son todos muy similares [66]. Cabe mencionar, que el procedimiento de Bootstrap permite además calcular IC para otras estadísticas además de la media, muchas de las cuales no podrían determinarse por otro método.

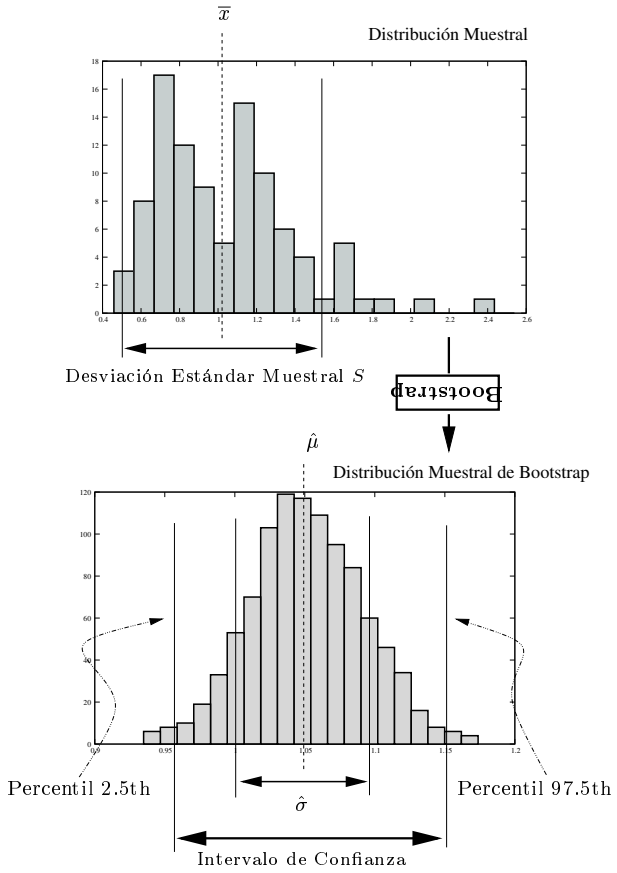


Figura 7.7: Cálculo de los intervalos de confianza

En la tabla 7.3 se muestran los intervalos de confianza percentiles para cada par **modelo-**

función con una confianza del 95 % para la media. De la tabla se observa que en la mayoría de los problemas los IC son muy reducidos (a excepción de $f05$), por lo que los valores mostrados en la tabla 7.2 son muy confiables como estimadores del comportamiento promedio del algoritmo. Por lo tanto, los tres modelos son capaces de resolver la mayoría de los problemas con un grado alto de confianza. En el caso del problema $f05$, los tres modelos logran resolverlo de manera consistente si se incrementa el número de evaluaciones.

Prob.	***rand/1/bin***	***best/1/bin***	***newde***
f01	***(0.0,0.0)***	***(0.0,0.0)***	***(0.0,0.0)***
f02	***(0.0,0.0)***	***(0.0,0.0)***	***(0.0,0.0)***
f04	*(0.9104,1.9237)*	*(0.0006,0.0025)*	*(0.00007,0.00009)*
f06	***(0.0,0.0)***	***(0.0,0.0)***	***(0.0,0.0)***
f07	***(0.0,0.0)***	***(0.0,0.0)***	***(0.0,0.0)***
f08	***(0.0,0.0)***	***(0.0,0.0)***	***(0.0,0.0)***
f09	***(0.0,0.0)***	***(0.0,0.0)***	*(0.139,0.358)*
f03	*(0.0232,0.0373)*	***(0.0,0.0)***	***(0.0,0.0)***
f10	***(0.0,0.0)***	***(0.0**,0.069309)*	***(0.0,0.0)***
f05	*(14.27,20.72)*	*(30.60,44.23)*	*(3.079,6.341)*
f11	*(0.0009,0.0038)*	*(0.0002,0.0017)*	***(0.0**,0.0004)*
f12	***(0.0,0.0)***	***(0.0,0.0)***	***(0.0,0.0)***
f13	***(0.0,0.0)***	***(0.0,0.0)***	***(0.0,0.0)***

Tabla 7.3: IC con 95 % de confiaza para la estadística media, en cada uno de los pares **modelo-problema**

7.1.7. Gráficas de Crecimiento

De las aptitudes promedio para cada par **modelo-función** (tabla 7.2) y de los IC para la estadística media (tabla 7.3), no es posible apreciar un comportamiento claramente diferente entre los tres modelos.

Se requieren diferentes cantidades de evaluaciones de la función objetivo para resolver cada función. Emplear un único valor de evaluaciones (en este caso $120,000$) en todas las funciones de prueba evita observar posibles diferencias entre los modelos. Por ejemplo, en la tabla 7.2 se observa que los tres modelos resuelven de manera consistente a las funciones $f01$, $f02$, $f06$, $f07$, $f08$, $f12$ y $f13$, pero en la tabla no muestra nada acerca de la cantidad real de evaluaciones necesarias para resolver cada función. Es decir, es posible que algún modelo emplee una cantidad reducida de evaluaciones a comparación de lo otros, pero esto no se puede apreciar en los resultados de la tabla.

Inicialmente se consideraron instancias de 30 variables de decisión para las trece funciones de prueba, como se propone en [92]. Dado que los problemas de prueba son insuficientes para poder establecer la existencia de una diferencia de comportamiento entre los tres modelos, se prosiguió a incrementar el número de variables de decisión de cada problema.

En la tabla 7.4 se muestra el comportamiento promedio (aptitud media de Bootstrap) de todos los pares **modelo-función** en instancias de problemas de 200 variables de decisión, manteniendo los mismos parámetros incluyendo el número de evaluaciones. De la tabla se aprecia una diferencia significativa en la dificultad de las funciones de prueba; por ejemplo, en la función $f03$ los tres algoritmos mostraron un comportamiento muy pobre empleando únicamente $120,000$ evaluaciones; algo similar sucede con $f04$ y $f05$. Por otro lado, en la función $f07$ los tres algoritmos obtuvieron resultados relativamente cercanos al óptimo considerando que se emplearon los mismos parámetros en instancias de 200 variables de decisión.

Prob.	***rand/1/bin***	***best/1/bin***	***newde***
f01	*294.336*	*5090.504**	***0.000005***
f02	*20.960925**	*11.055**	***0.001455***
f04	***39.823***	*47.562*	*65.93*
f06	*291.0251**	*35.398**	***0.0***
f07	*0.2185*	*6.0280**	***0.0***
f08	*-64952.92*	*-65170.59*	***-70925.08***
f09	*302.012**	*294.7123**	***3.4447***
f03	*199250.930**	*57584.865*	***45283.67***
f10	*5.8206**	*3.4085**	***0.000486***
f05	*87531.83**	*809.006*	***667.57***
f11	*3.4710**	*0.0102*	***0.000632***
f12	*2.5833**	*5.9987**	***0.0***
f13	*11833.31**	*92.94**	***0.000001***

Tabla 7.4: Comportamiento promedio para cada par **modelo-función** empleando instancias de 200 variables de decisión

Lo realmente importante de la tabla no es si los algoritmos encuentran la solución de manera consistente en cada función, puesto que lo hacen si se incrementa el número de evaluaciones, sino la relación que guardan las aptitudes medias en cada función para cada modelo. Por ejemplo, en el caso de $f01$, el modelo **newde** obtuvo un comportamiento promedio cercano al óptimo (0.00005), mientras que los modelos **rand/1/bin** y **best/1/bin** obtuvieron una aptitud promedio 5.8×10^7 y 1.0×10^9 veces más que el modelo **newde**, respectivamente. Es decir, las diferencias en la aptitud promedio es de varios órdenes de

magnitud. Se observa además, que estas diferencias en órdenes de magnitud entre aptitudes promedio ocurren en otras funciones además de $f01$, por ejemplo: $f02$, $f06$, $f07$, $f09$, $f10$, $f12$, $f13$ (véase las filas sombreadas. Para valores de **0.0** se asegura que al menos se obtuvo una precisión de 1×10^{-7} con respecto al óptimo).

Así mismo, en todos los casos, a excepción de $f04$, el modelo **newde** obtiene claramente mejores aptitudes promedio que los otros dos modelos. Además, cabe mencionar que la diferencia entre los modelos para la función $f04$ no es de órdenes de magnitud. En todas las funciones de prueba, a excepción de $f08$, el valor óptimo no cambia al incrementarse las variables de decisión. De la tabla 7.4 se observa que las aptitudes promedio de los modelos para $f08$ son negativas ya que no se sabe cuál es el valor óptimo global para esta instancia del problema.

Con el fin de caracterizar el comportamiento de la aptitud promedio para cada par **modelo-función** conforme se incrementa el número de variables de decisión del problema, se efectuó un conjunto de 100 ejecuciones independientes por cada modelo en instancias de 100, 200, 300 y 400 variables de decisión para cada función de prueba. En este estudio no se incluyó a $f08$ dado que no se sabe su valor óptimo para tales instancias.

En las tablas 7.5 y 7.6 se muestran las gráficas del comportamiento de la aptitud promedio conforme se incrementa el número de variables de decisión del problema manteniendo constante todos los parámetros para cada uno de los modelos. Las gráficas se muestran en escala logarítmica dada la diferencia de magnitud entre las aptitudes promedio. Para el modelo **newde** algunas de las gráficas no están definidas en todas las instancias de las funciones, esto porque el logaritmo de cero no está definido.

Se observa que la aptitud promedio del modelo **newde** en la función $f04$ es la peor para todas las instancias del problema pero éste es el único caso donde ocurre tal comportamiento. En todas las demás gráficas se aprecia que en todas las instancias de las funciones el modelo **newde** muestra un mejor comportamiento promedio. En los casos: $f01$, $f02$, $f05$, $f06$, $f07$, $f09$, $f10$, $f12$ y $f13$ la diferencia es de varios órdenes de magnitud. Por otro lado, el modelo **best/1/bin** muestra un comportamiento ligeramente mejor que el modelo tradicional **rand/1/bin**.

Con la finalidad de caracterizar el comportamiento promedio del algoritmo en todas la funciones de prueba, se normalizó cada una de las gráficas de las tablas 7.5 y 7.6 en el intervalo $[0, 1]$, esto debido a que los rangos de la aptitud promedio son dependientes del problema. Al normalizar las gráficas se mantiene la relación que guardan los tres modelos para cada función. Es decir, se busca escalar las proporciones que guardan los modelos entre sí para que sean comparables a lo largo de todas las funciones de prueba.

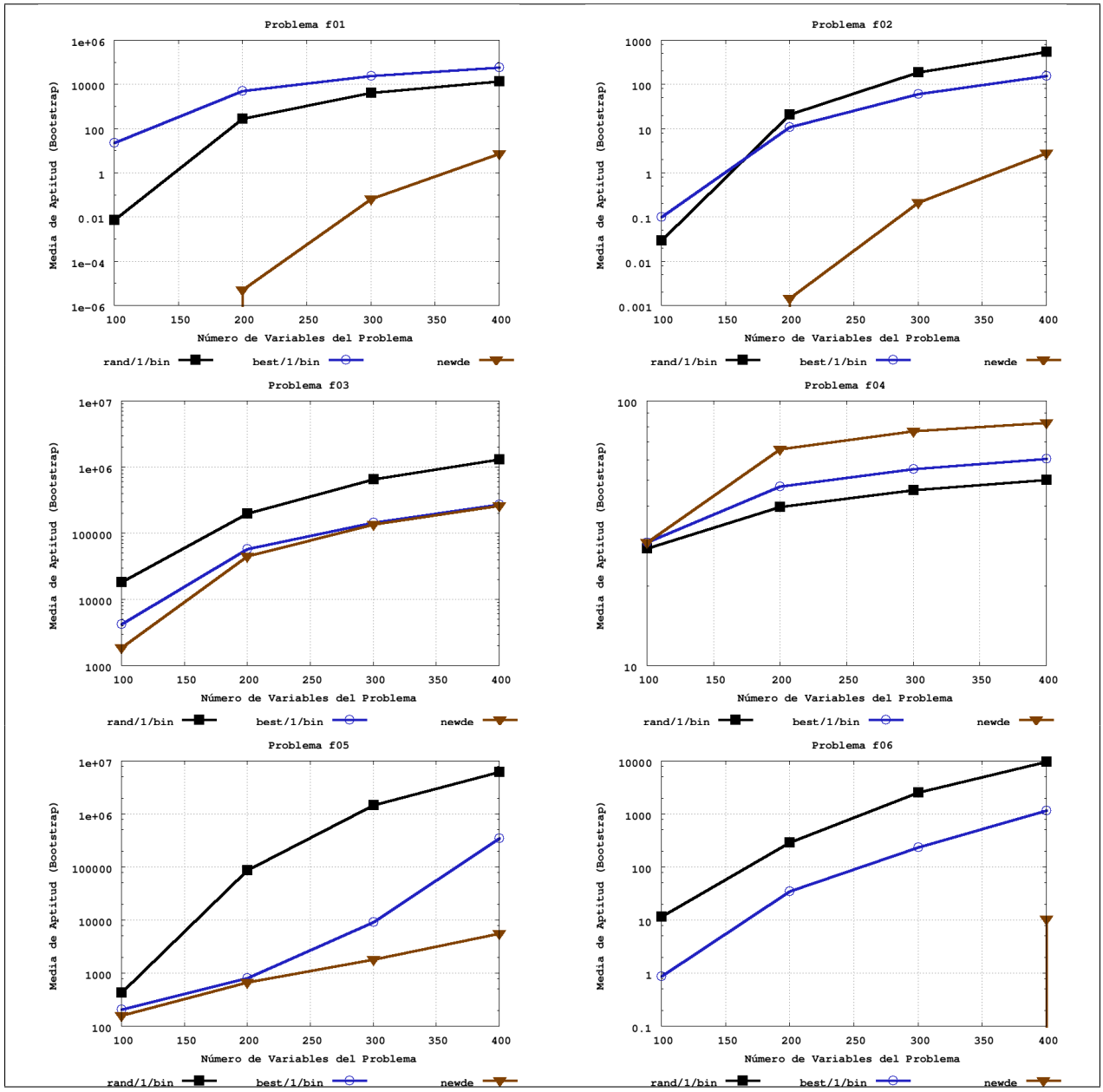


Tabla 7.5: Gráficas de la Aptitud Media vs. Número de Variables de Decisión, para las funciones: $f01, f02, f03, f04, f05, f06$. La Aptitud Media es calculada empleando un procedimiento de bootstrap con 1000 remuestreos para un conjunto de resultados de 100 ejecuciones independientes por cada uno de los modelos: **rand/1/bin, best/1/bin, newde**.

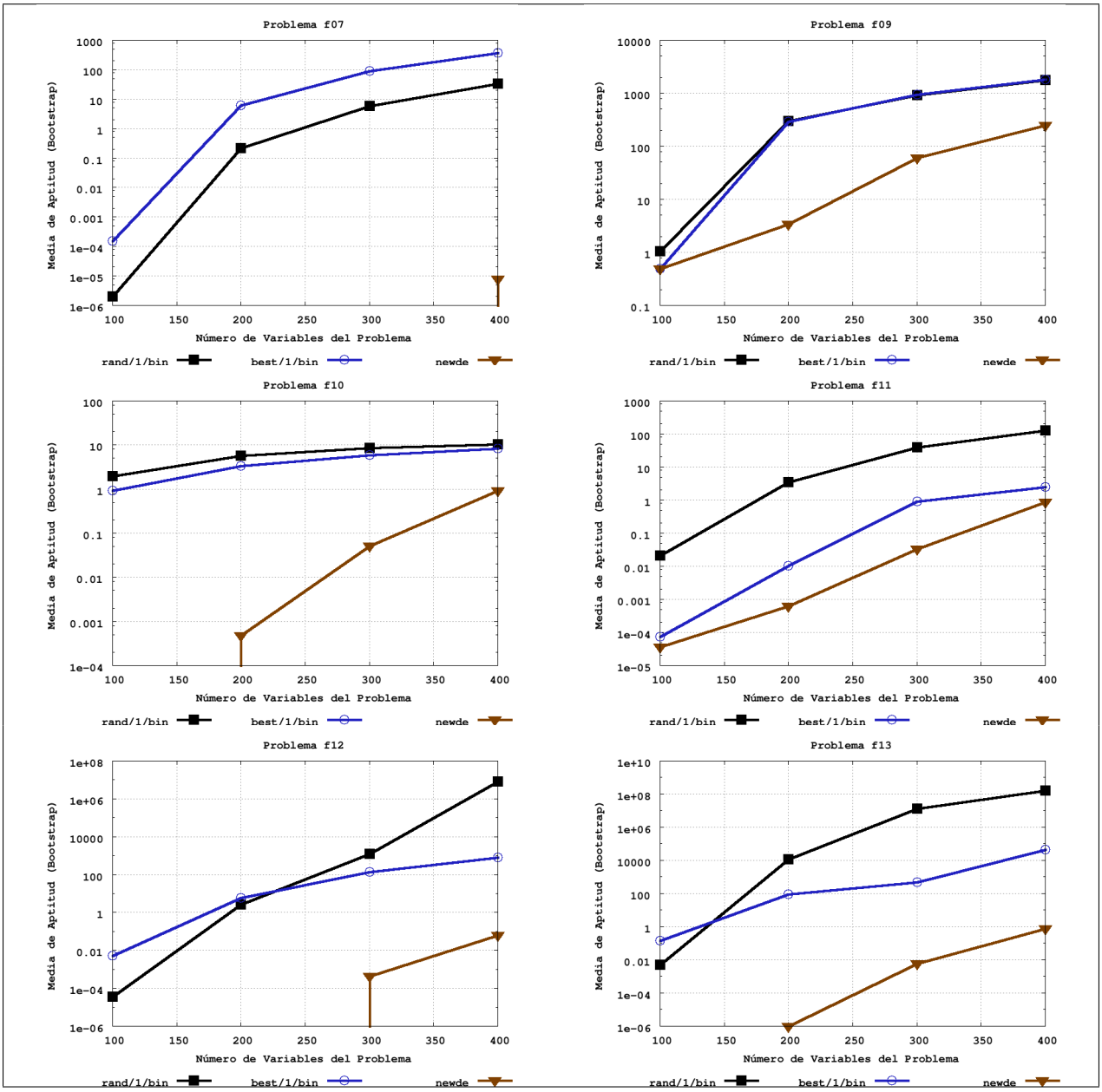


Tabla 7.6: Gráficas de la Aptitud Media vs. Número de Variables de Decisión, para las funciones $f07, f09, f10, f11, f12, f13$. La Aptitud Media es calculada empleando un procedimiento de bootstrap con 1000 remuestreos para un conjunto de resultados de 100 ejecuciones independientes por cada uno de los modelos: **rand/1/bin, best/1/bin, newde**.

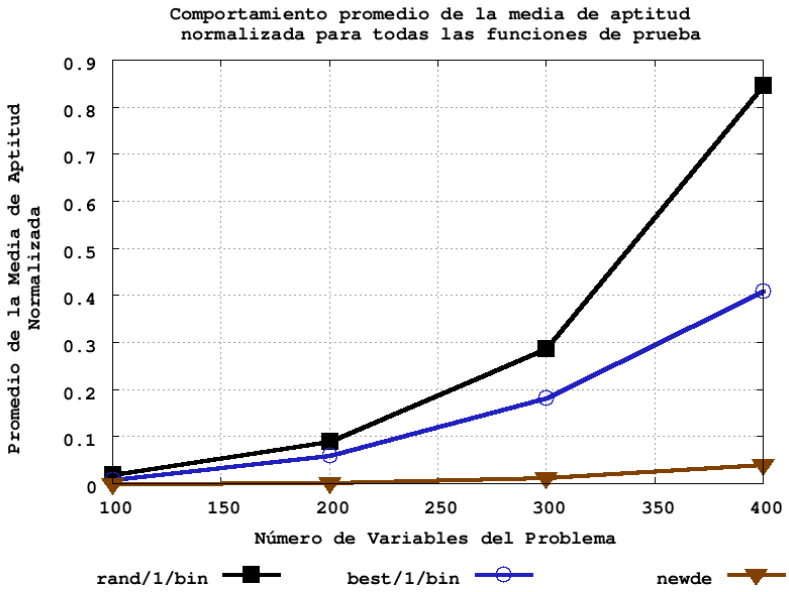


Figura 7.8: Comportamiento promedio global de los tres modelos

En la gráfica 7.8 se muestra el resultado promedio de todas las gráficas normalizadas. La gráfica promedio por cada modelo se obtuvo empleando un procedimiento de Bootstap. En esta gráfica quedan muy claras las diferencias de comportamiento entre los tres modelos. Se observa que conforme se incrementa el número de variables de decisión, las diferencias entre las aptitudes promedio del modelo **rand/1/bin** con respecto a los otros modelos crecen en órdenes de magnitud. Lo mismo sucede al compararse **best/1/bin** y **newde**. De la gráfica se concluye que el modelo **newde** mejora de manera significativa el comportamiento del algoritmo de Evolución Diferencial a comparación del modelo tradicional y más ampliamente usado **rand/1/bin** así como al modelo **best/1/bin** que mostró el mejor comportamiento de entre otros modelos propuestos al algoritmo de Evolución Diferencial.

Hasta ahora los parámetros empleados han sido los mismos en todos los casos. En particular, el número de evaluaciones de la función objetivo se ha mantenido constante. En la gráfica 7.9 se muestra el número de evaluaciones necesarias para resolver el problema del elipsoide con una precisión de 1.0×10^{-6} para los modelos **rand/1/bin** y **newde**, en este caso se cambió el criterio de terminación del algoritmo a un valor de tolerancia $\epsilon = 1.0 \times 10^{-6}$ con respecto al valor óptimo; los demás parámetros se mantuvieron constantes (tamaño de población μ, coeficientes de recombinación CR, y número de descendientes λ). Únicamente

se comparan estos modelos, ya que el modelo **best/1/bin** no pudo resolver el problema para instancias mayores a 200 variables con la precisión deseada. Era necesario un cambio en el tamaño de la población ya que sufría de *estancamiento* (esto a consecuencia de emplear siempre como inicio de trayectoria a la mejor solución en la población).

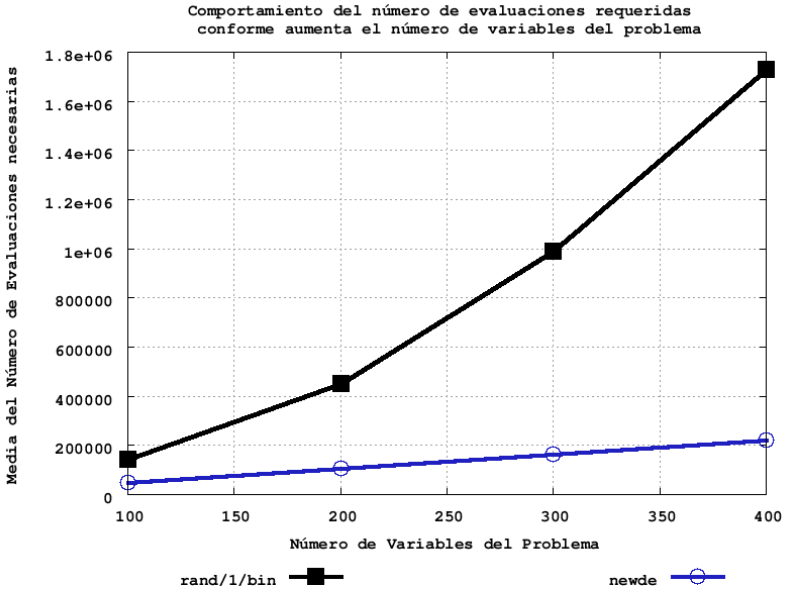


Figura 7.9: Rand vs Newde

En la gráfica 7.9 se aprecia una clara diferencia en la cantidad de evaluaciones necesarias para resolver el problema del elipsoide, algo muy similar al comportamiento de la aptitud promedio mostrado en la figura 7.8. De hecho, se observa una diferencia de orden en el crecimiento del número de evaluaciones conforme se incrementa el número de variables del problema.

Para el caso del modelo **newde** se consideró una función de crecimiento lineal de la forma $m \cdot x + b$. Empleando el método de mínimos cuadrados se ajustó la curva a los datos de la gráfica 7.9. Después, se extrapoló esta curva con el fin de predecir el número de evaluaciones necesarias para instancias mayores a 400 variables de decisión. Con el fin de validar esta aproximación, se calculó (usando 100 ejecuciones independientes) el número promedio de evaluaciones de la función objetivo para una instancia del problema de 2000 variables de decisión. La diferencia encontrada entre el valor promedio real y el esperado por el modelo lineal fue únicamente de 6 %. Considerando que se extrapoló a una instancia del problema

500 % mayor que los datos calculados, se acepta como buena aproximación el modelo lineal. Estos resultados se muestran en la figura 7.10.

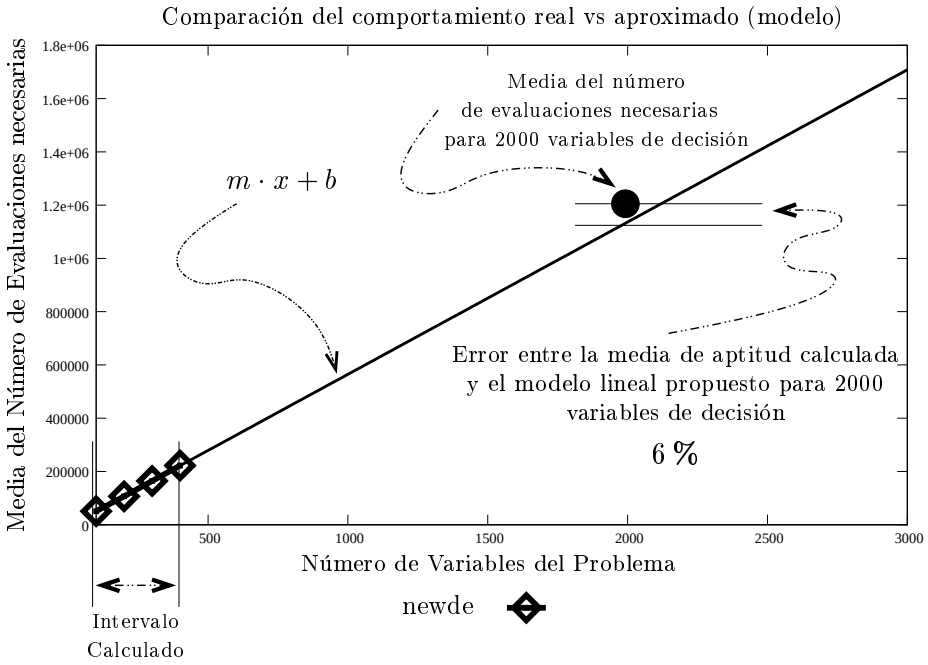


Figura 7.10: Aproximación del comportamiento del modelo **newde**

Por otro lado, para el modelo **rand/1/bin** se consideró una función de crecimiento cuadrático de la forma $a_2 \cdot x^2 + a_1 \cdot x + a_0$. Empleando el método de mínimos cuadrados se ajustó la curva a los datos calculados. Con la finalidad de validar esta aproximación, se calculó (usando 100 ejecuciones independientes) el número promedio de evaluaciones de la función objetivo para la instancia del problema de 600 variables. Esto difiere del caso anterior, principalmente por la cantidad tan grande de evaluaciones necesarias para instancias mayores así como la insuficiencia de la representación de punto flotante empleada. La diferencia encontrada entre el valor promedio real y el esperado por el modelo cuadrático fue únicamente del 2 %, con lo cual se acepta como buena aproximación el modelo cuadrático. Estos resultados se muestran en la gráfica 7.11.

Finalmente, en la gráfica 7.12 se muestra una comparativa de los modelos de crecimiento lineal y cuadrático determinados para los modelos **newde** y **rand/1/bin** respectivamente. La gráfica se presenta en escala logarítmica debido a la diferencia de magnitudes entre las aproximaciones. Para 600 variables de decisión existe una diferencia de 11 veces el número

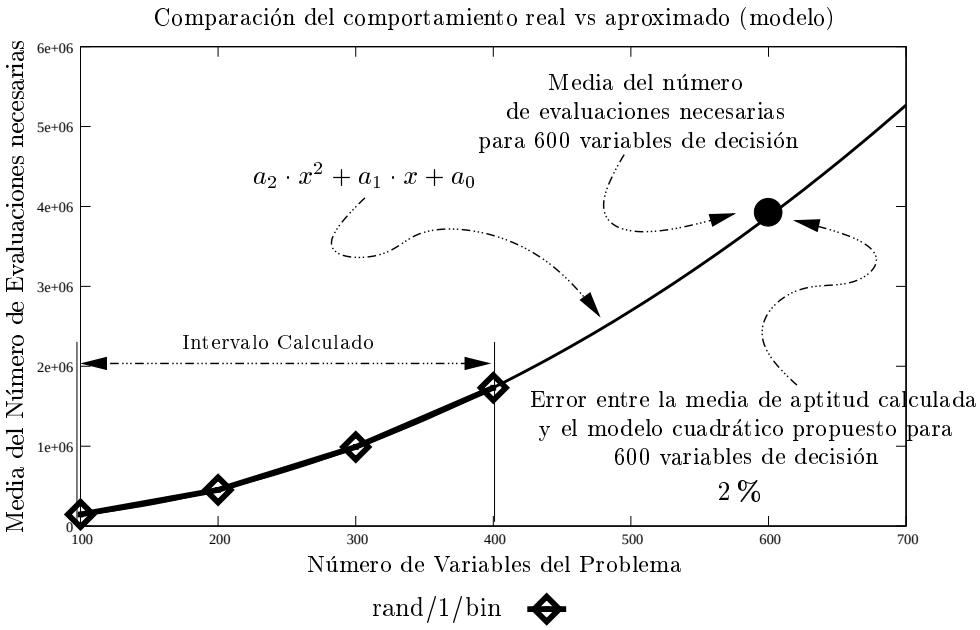


Figura 7.11: Aproximación del comportamiento del modelo **rand/1/bin**

de evaluaciones requeridas, mientras que para 2000 variables, la diferencia es de 38 veces. Esta diferencia aumenta conforme se incrementa el número de variables de decisión.

Por último, se concluye que existe una diferencia de orden de crecimiento en el número de evaluaciones necesarias para resolver el problema del elipsoide. Esta diferencia de comportamiento no es exclusiva para el problema $f01$. De los resultados mostrados en las tablas 7.5, 7.6 se espera un comportamiento similar en varias de las funciones empleadas algunas de las cuales cuentan con características de multimodalidad y no separabilidad, siendo éstas las funciones más difíciles de resolver.

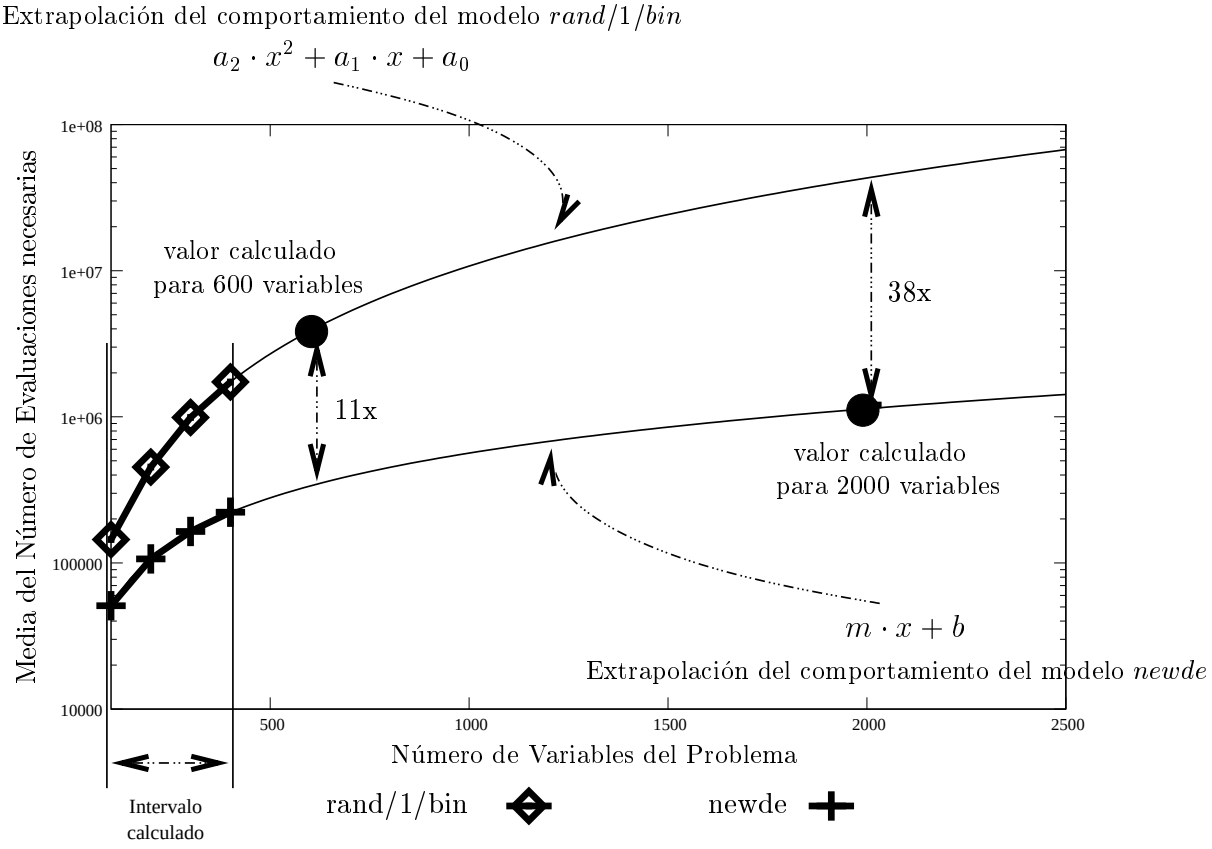


Figura 7.12: Comparación de la escalabilidad de los modelos **rand/1/bin** y **newde**

7.2. Propuesta para Optimización Restringida

Con la finalidad de evaluar el desempeño de la propuesta de Evolución Diferencial para optimización numérica con restricciones se empleó un conjunto de trece problemas de prueba estándares en la literatura de optimización restringida empleando algoritmos evolutivos; estos problemas se listan en el Apéndice B al final de este documento.

Para estimar la dificultad de generar una solución factible para cada uno de estos problemas se emplea la métrica ρ propuesta por Michalewicz y Schoenauer [64]. Esta métrica pretende estimar la porción de la zona factible con respecto a todo el espacio de búsqueda, así $\rho = \frac{|F|}{|S|}$ donde $|F|$ y $|S|$ son aproximaciones al tamaño de la zona factible $\mathcal{F}$ y al tamaño del espacio de búsqueda $\mathcal{S}$. Para calcular $|F|$ y $|S|$ se generan $|S| = 1,000,000$ de soluciones uniformemente distribuidas en todo el espacio de búsqueda y se determina el número de soluciones factibles $|F|$. En la tabla 7.2 se listan algunas de las características de los problemas de prueba incluyendo los valores de ρ, el número de variables de decisión n y el tipo de función objetivo. De la tabla se puede apreciar que los problemas poseen una amplia variedad de características.

problema	n	$f(\vec{x})$	ρ
g01	13	cuadrática	0.0003 %
g02	20	no lineal	99.9973 %
g03	10	no lineal	0.0026 %
g04	5	cuadrática	27.0079 %
g05	4	no lineal	0.0000 %
g06	2	no lineal	0.0057 %
g07	10	cuadrática	0.0000 %
g08	2	no lineal	0.8581 %
g09	7	no lineal	0.5199 %
g10	8	lineal	0.0020 %
g11	2	cuadrática	0.0973 %
g12	3	cuadrática	4.7697 %
g13	5	no lineal	0.0000 %

Además, la nueva propuesta (**newde**) se comparó contra los siguientes algoritmos: **CDE** *Constrained Differential Evolution* propuesto por Lampinen [55], el cual representa a la mejor propuesta para optimización restringida empleando Evolución Diferencial, **SMES** *Simple Multimember Evolution Strategy* propuesto por Mezura [60], la cual emplea un mecanismo similar de manejo de restricciones e **ISRES** *Improved Stochastic Ranking Evolution Strategy* propuesto por Runnarson y Yao [81], ésta es la mejor propuesta en la literatura de computación evolutiva para optimización numérica restringida.

Se emplearon las versiones en línea disponibles por los autores de los algoritmos **SMES** e **ISRES**; este último fue implementado en el lenguaje de cómputo científico Matlab ®, por lo que se convirtió al lenguaje C para evitar diferencias de comportamiento a consecuencia de la generación de números aleatorios o de la representación de punto flotante empleada. La versión de **ISRES** en lenguaje C mostró los mismos resultados que los reportados en su fuente original [81]. Por otro lado, los autores de **CDE** no presentan una versión disponible en línea de su algoritmo, por lo que se tuvo que implementar siguiendo la descripción mostrada en [55]. Los resultados obtenidos con la implementación en lenguaje C concuerdan con los reportados en su fuente original [55].

7.2.1. Parámetros de Control Empleados

Para cada algoritmo se emplearon los parámetros mostrados en sus fuentes originales. Tanto para **ISRES** y **SMES** los resultados reportados emplean $240,000$ evaluaciones de la función objetivo; por otro lado, para **CDE** en su fuente original se emplea un número variable de evaluaciones de la función objetivo para cada problema así como diferentes tamaños de población. Por tal motivo se empleó como tamaño único de población al tamaño máximo reportado en [55]. Además se ajusto el número de generaciones para que el número de evaluaciones fuera $240,000$.

	μ	λ	G_{max}	cf
ISRES	60	400	600	no aplica
SMES	100	300	800	no aplica
CDE	120	no aplica	2000	0.9
newde	30	150 (5)	1600	0.9

Tabla 7.7: Parámetros empleados en las pruebas experimentales. Donde μ es el tamaño de la población, λ es el número de descendientes, G_{max} es el número de generaciones y cf es el coeficiente empleado únicamente por la ED

En el caso de la propuesta **newde** el número de descendientes $\lambda = 150$ implica que se generan cinco descendientes por cada solución dentro de la población, además los valores de F_α y F_β son los mismos que para el caso sin restricciones. Para todos los algoritmos el número de evaluaciones de la función objetivo es de $240,000$.

7.2.2. Valores de Aptitud Media e Intervalos de Confianza

Con la finalidad de determinar el comportamiento promedio de los cuatro algoritmos, se

realizó un conjunto de cien ejecuciones independientes por cada par **algoritmo-problema**. En la figura 7.13 se muestra un histograma de frecuencias del algoritmo **newde** para el problema $g02$ donde se observa que los resultados no se ajustan a una distribución normal.

En el Apéndice C.1 se muestran los histogramas de frecuencias de las soluciones para cada uno de los problemas de prueba para el algoritmo **newde**, de los cuales se observa que en ninguno de los casos los datos se ajustan a una forma normal. Además, este comportamiento no es propio de la propuesta **newde** sino que también está presente en los demás algoritmos. Por tal motivo, al igual que en el caso de optimización sin restricciones, se empleó un procedimiento de Bootstrap con 1000 remuestreos para estimar el valor de la media de la población a la cual pertenecen las cien ejecuciones independientes.

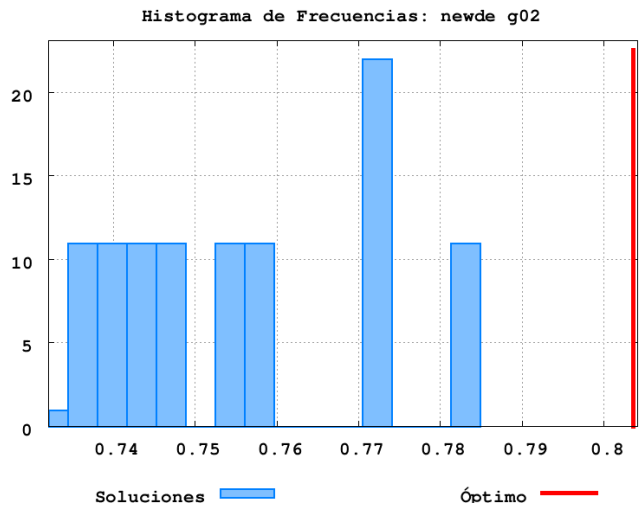


Figura 7.13: Histogramas de Frecuencias de las soluciones del algoritmo propuesto **newde** para el problema $g02$. Se observa que la muestra no se ajusta a una forma normal

En la tabla 7.8 se muestran los valores de la aptitud promedio (Bootstrap) para cada uno de los algoritmos en todos los problemas de prueba. Un valor en **negritas** indica el mejor resultado obtenido, mientras que los valores marcados con * indican que el valor medio no concuerda con el valor óptimo.

De la tabla 7.8 se observa que el algoritmo propuesto **newde** obtiene los mejores resultados para los trece problemas de prueba. Únicamente en las funciones $g02$, $g03$, $g07$, $g11$ y $g13$ la media de aptitud no concuerda con el valor óptimo, pero cabe observar que en todos estos casos la media de aptitud está muy cercana al óptimo. En todos los demás problemas el valor óptimo se obtiene de manera consistente.

Por otro lado, el algoritmo **ISRES** muestra resultados competitivos en comparación con la propuesta **newde**; en diez de los trece problemas obtiene los mismos resultados que **newde**. Con respecto al algoritmo **CDE**, éste logra resolver seis de los trece problemas de prueba de manera consistente pero, en la mayoría de los problemas con un valor pequeño de ρ el algoritmo muestra muy pobres resultados; tal es el caso de los problemas $g03$, $g05$,

$g07$, $g10$ y $g13$. Por último, el algoritmo **SMES** únicamente logra resolver consistentemente a tres de los trece problemas de prueba; sin embargo, muestra un comportamiento más estable que **CDE**, es decir, proporciona buenas aproximaciones al valor óptimo en la mayoría de los casos. De la tabla 7.8 se aprecia que los algoritmos **ISRES** y **newde** son los más competitivos, resolviendo de manera consistente la mayoría de los problemas.

	óptimo	**cde**	**smes**	**isres**	**newde**
g01 (min)	-15	-13.93461	-14.98877	-14.97545	**-15.00000**
g02 (max)	0.803619	0.41368	0.78447	0.76330	**0.80309***
g03 (max)	1	0.24616	1.00082	**1.00050**	**1.00050***
g04 (min)	-30665.539	**-30665.53867**	**-30665.53867**	**-30665.53867**	**-30665.53867**
g05 (min)	5126.4981	5315.59999	5198.13409	**5126.49671**	**5126.49671**
g06 (min)	-6961.81388	**-6961.81388**	-6954.62568	**-6961.81388**	**-6961.81388**
g07 (min)	24.3062091	24.78596	24.49414	24.30645	**24.30638***
g08 (min)	0.095825	**0.09583**	**0.09583**	**0.09583**	**0.09583**
g09 (min)	680.6300573	**680.63006**	680.64977	**680.63006**	**680.63006**
g10 (min)	7049.3307	7090.50762	7255.29878	**7049.24842**	**7049.27286**
g11 (min)	0.75	**0.74990**	0.74734	**0.74990**	**0.74990***
g12 (max)	1	**1.0**	**1.0**	**1.0**	**1.0**
g13 (min)	0.0539498	0.80852	0.16613	0.08139	**0.05784***

Tabla 7.8: Aptitud Media calculada empleando un procedimiento de Bootstrap con 1000 remuestreos

En las tablas 7.9 y 7.10 se presentan los intervalos de confianza para la media de aptitud de cada par **algoritmo-problema**. Estos resultados concuerda con los de la tabla 7.8, donde el algoritmo **newde** muestra los mejores resultados. De la tabla 7.10, se observa que en todos lo problemas a excepción de $g02$ y $g13$, los intervalos de confianza de **newde** contienen al valor óptimo, y en caso contrario están muy próximos a él. Además, se aprecia también que **CDE** obtiene muy malos resultados en siete de trece funciones de prueba. Por ejemplo, para el problema $g05$ no sólo el intervalo de confianza es muy amplio sino que además está muy lejos del valor óptimo. Finalmente, el algoritmo **SMES** únicamente obtiene intervalos que contienen al valor óptimo en cuatro de los trece problemas de prueba.

De las tablas 7.8, 7.9 y 7.10 se observa que la propuesta **newde** muestra los mejores resultados pero, emplear un único valor de evaluaciones (en este caso $240,000$) en todos los problemas de prueba evita observar otras posibles diferencias entre los algoritmos, además de que cada problema requiere una cantidad diferente de evaluaciones para ser resuelto. A diferencia de los problemas de prueba sin restricciones, los problemas de prueba para optimización con restricciones no son escalables en el número de variables de decisión. Por tal motivo, se redujo el número de evaluaciones de la función objetivo con la finalidad de

	óptimo	cde	smes
g01 (min)	-15	(-13.961,-13.906)	(-15.000,-14.953)
g02 (max)	0.803619	(0.408,0.419)	(0.780,0.788)
g03 (max)	1	(0.223,0.271)	**(1.000,1.001)**
g04 (min)	-30665.539	**(-30665.539,-30665.539)**	**(-30665.539,-30665.539)**
g05 (min)	5126.4981	(5276.757,5356.311)	(5180.455,5218.104)
g06 (min)	-6961.81388	**(-6961.814,-6961.814)**	(-6961.763,-6942.908)
g07 (min)	24.3062091	(24.765,24.812)	(24.465,24.526)
g08 (min)	0.095825	**(0.096,0.096)**	**(0.096,0.096)**
g09 (min)	680.6300573	**(680.630,680.630)**	(680.644,680.659)
g10 (min)	7049.3307	(7088.781,7092.395)	(7229.808,7282.667)
g11 (min)	0.75	**(0.750,0.750)**	(0.743,0.749)
g12 (max)	1	**(1.0,1.0)**	**(1.0,1.0)**
g13 (min)	0.0539498	(0.738,0.887)	(0.134,0.206)

Tabla 7.9: Intervalos de Confianza de los trece problemas de prueba para los algoritmos **CDE** y **SMES**

	óptimo	isres	newde
g01 (min)	-15	(-15.000,-14.924)	**(-15.000,-15.000)**
g02 (max)	0.803619	(0.757,0.769)	**(0.803,0.803)***
g03 (max)	1	(1.001,1.001)	**(1.000,1.000)**
g04 (min)	-30665.539	**(-30665.539,-30665.539)**	**(-30665.539,-30665.539)**
g05 (min)	5126.4981	**(5126.497,5126.497)**	**(5126.497,5126.497)**
g06 (min)	-6961.81388	**(-6961.814,-6961.814)**	**(-6961.814,-6961.814)**
g07 (min)	24.3062091	**(24.306,24.307)**	**(24.306,24.306)**
g08 (min)	0.095825	**(0.096,0.096)**	**(0.096,0.096)**
g09 (min)	680.6300573	**(680.630,680.630)**	**(680.630,680.630)**
g10 (min)	7049.3307	**(7049.248,7049.249)**	**(7049.267,7049.279)**
g11 (min)	0.75	**(0.750,0.750)**	**(0.750,0.750)**
g12 (max)	1	**(1.000,1.000)**	**(1.000,1.000)**
g13 (min)	0.0539498	(0.065,0.100)	**(0.054,0.065)***

Tabla 7.10: Intervalos de Confianza de los trece problemas de prueba para los algoritmos **ISRES** y **newde**

observar el comportamiento de los algoritmos durante el proceso de búsqueda.

En la tabla 7.11 se muestran los valores de aptitud promedio de cada conjunto de 100 ejecuciones independientes para cada par **algoritmo-problema** empleando 24,000 evaluaciones de la función objetivo (10 % de las evaluaciones empleadas en [60, 81]). Los valores en **negritas** indican el mejor valor encontrado para los cuatro algoritmos y los valores marcados con * indican que no se obtuvo el valor óptimo consistentemente. De la tabla no sólo se observa que el algoritmo **newde** obtiene los mejores resultados en once de los trece problemas de prueba, sino que además, en las funciones $g04$, $g05$, $g06$, $g08$, $g09$ y $g12$ encuentra el valor óptimo consistentemente. También se observa que existe una notable diferencia entre los resultados de **newde** y los demás algoritmos incluyendo a **ISRES**. Por ejemplo, obsérvese los resultados para los problemas $g01$, $g05$, $g07$ y $g10$.

	óptimo	**cde**	**smes**	**isres**	**newde**
g01 (min)	-15	-4.92985	-14.42362	-7.75971	**-14.84763***
g02 (max)	0.803619	0.29223	0.70887	0.65611	**0.75751***
g03 (max)	1	0.06494	**0.95560***	0.82655	0.81203
g04 (min)	-30665.539	-30641.42061	-30661.27170	-30660.77686	**-30665.53867**
g05 (min)	5126.4981	5176.21101	5198.11211	5147.74175	**5126.49671**
g06 (min)	-6961.81388	-6961.81293	-6905.81826	-6910.98997	**-6961.81388**
g07 (min)	24.3062091	1128.83538	26.50418	29.47595	**24.36134***
g08 (min)	0.095825	**0.09583**	**0.09583**	**0.09583**	**0.09583**
g09 (min)	680.6300573	687.90481	681.23900	681.77135	**680.63006**
g10 (min)	7049.3307	19688.97638	8316.38788	8064.13421	**7124.78133***
g11 (min)	0.75	0.89610	0.75270	**0.74990**	**0.74990**
g12 (max)	1	**1.0**	**1.0**	**1.0**	**1.0**
g13 (min)	0.0539498	0.67603	**0.06360***	0.08375	0.36538

Tabla 7.11: Aptitud Media calculada a través de un procedimiento de Bootstrap con 1000 remuestreos. En cada ejecución se emplearon 24,000 evaluaciones

7.2.3. Gráficas de Crecimiento de la Aptitud Media

Con la finalidad de apreciar el comportamiento de los algoritmos durante el proceso de búsqueda se generó un conjunto de 100 ejecuciones independientes por cada par **algoritmo-problema** para diferente número de evaluaciones. Es decir, se ajustó el número de generaciones en cada algoritmo para un número dado de evaluaciones mientras los demás parámetros se mantuvieron constantes. En el experimento se emplearon diez diferentes niveles de evaluaciones, correspondiendo cada uno al $10\,\%, 20\,\%, \ldots, 100\,\%$ de las 240,000 evaluaciones empleadas en [60, 81].

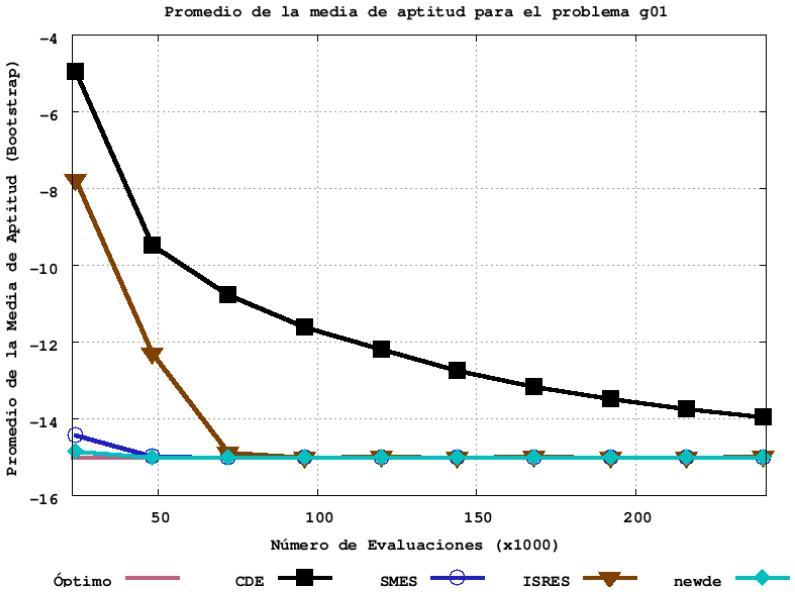


Figura 7.14: Comportamiento de la Aptitud Media conforme se incrementa el número de evaluaciones de la función objetivo para el problema $g01$

En la gráfica 7.14 se muestra el comportamiento de la aptitud media de cada uno de los algoritmos conforme se incrementa el número de evaluaciones de la función objetivo, esto para el problema $g01$. Se observa de la figura que para los algoritmos **CDE** y **SMES** se mejora la aptitud promedio conforme se incrementa el número de evaluaciones; mientras que para el modelo **ISRES** el algoritmo llega muy cerca del valor óptimo después de $50,000$ evaluaciones. Por otro lado, el algoritmo **newde** encuentra la zona del valor óptimo inmediatamente aún con únicamente $24,000$ evaluaciones.

En las figuras 7.15 y 7.16 se muestran las gráficas de comportamiento de la aptitud media. Para los doce problemas de prueba restantes. De las gráficas se observa que para los problemas $g02$, $g03$, $g05$ y $g13$ el algoritmo **CDE** muestra un muy mal desempeño, para los demás problemas se aprecia que es este algoritmo el que requiere mayor número de evaluaciones para encontrar la zona donde se localiza el valor óptimo. Esto se debe principalmente a su esquema de selección que únicamente se basa en la factibilidad de las soluciones y no establece ningún balance entre soluciones factibles e infactibles. Por otro lado, en las gráficas aún no se puede apreciar una clara diferencia entre los algoritmos **ISRES** y **newde**, esto debido principalmente a las grandes diferencias de aptitud entre el algoritmo **CDE** y las demás propuestas.

Un aspecto importante que se aprecia en la gráficas es que en todos los problemas de prueba a excepción de $g03$ y $g13$ el algoritmo **newde** se aproxima al valor óptimo de manera casi inmediata aún con sólo el 10 % de las 240, 000 evaluaciones. Esto debido al nuevo modelo que incorpora información de la trayectoria de cada solución así como información de la mejor solución en la población.

7.2.4. Distribución Muestral de la Distancia Promedio

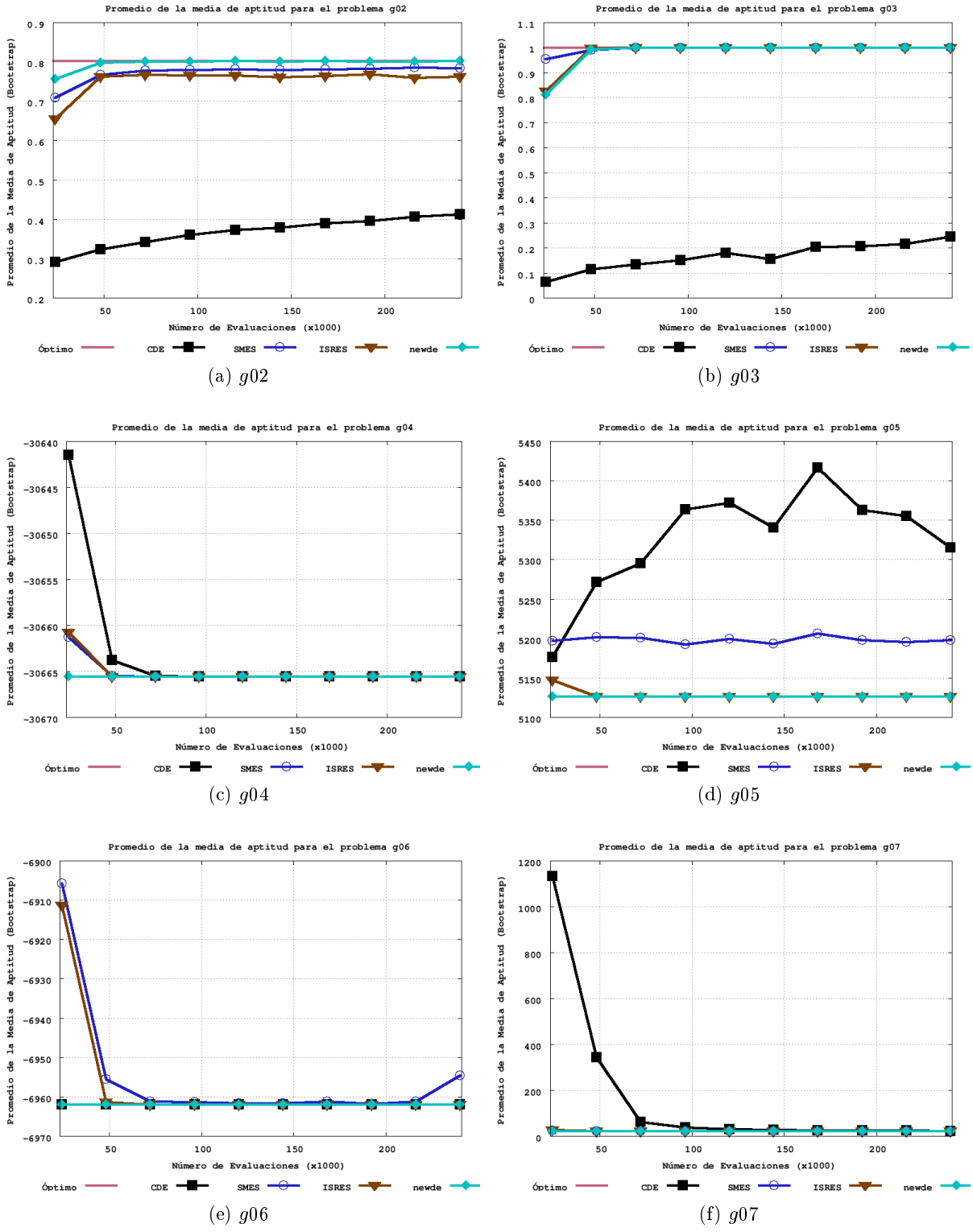
De las gráficas 7.15 y 7.16 se puede observar una diferencia significativa entre el algoritmo **CDE** y las demás propuestas. Sin embargo, aún no se puede observar un claro comportamiento diferente entre los algoritmos **SMES**, **ISRES** y **newde**.

Además de no ser escalables (excepto por $g02$), los problemas de prueba empleados en optimización con restricciones difieren tanto en su valor óptimo, como en la localización del mismo; en el caso de las funciones de prueba sin restricciones todas tienen un valor óptimo de cero lo que facilita la comparación de resultados. Por lo que no es posible obtener un comportamiento promedio para todos los problemas de prueba tal como se mostró en los resultados del nuevo modelo de ED (sección 7.1).

Una de las ventajas de emplear un procedimiento de Bootstrap es que es posible determinar distribuciones de probabilidad muestrales para diferentes estadísticas además de la media. Por tal motivo, se propone emplear una estadística que mida la proximidad de las soluciones al valor óptimo. Así, para una muestra $M = \{x_1, x_2, \ldots, x_n\}$ de tamaño n, la distancia promedio al óptimo $\hat{d}_p$ para el problema p está determinada por la ecuación (7.1). Donde x_p^* representa el valor óptimo de la función objetivo para el problema p.

$$\hat{d}_p(M) = \frac{\sqrt{\sum_{i=1}^{n}(x_i - x_p^*)^2}}{n} \tag{7.1}$$

Con el procedimiento de Bootstrap mostrado en el algoritmo 14 se calcula la **distribución muestral de la distancia promedio al valor óptimo** para cada población de soluciones. $\hat{d}_p$ permite medir la *precisión* del algoritmo, esto es, la proximidad de las soluciones al valor óptimo, además de que da un estimado de la dispersión de las mismas. Si el valor de $\hat{d}_p(M)$ es pequeño implica que la mayoría de las soluciones están cercanas al valor óptimo.

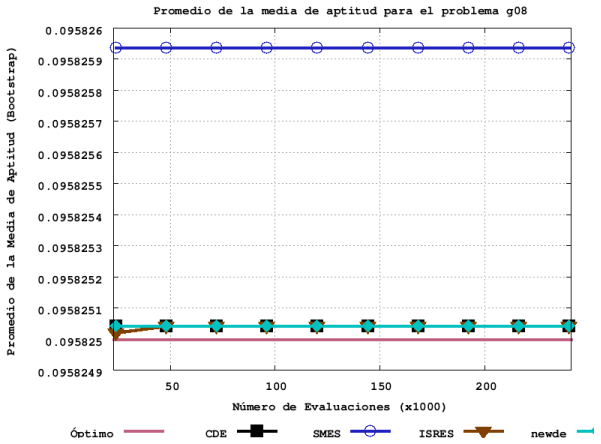


(a) $g02$ (b) $g03$

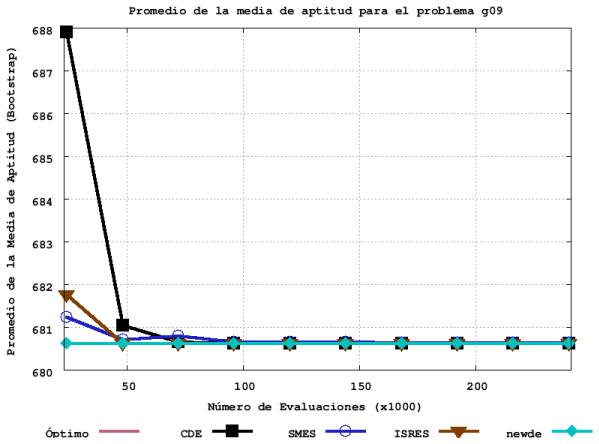
(c) $g04$ (d) $g05$

(e) $g06$ (f) $g07$

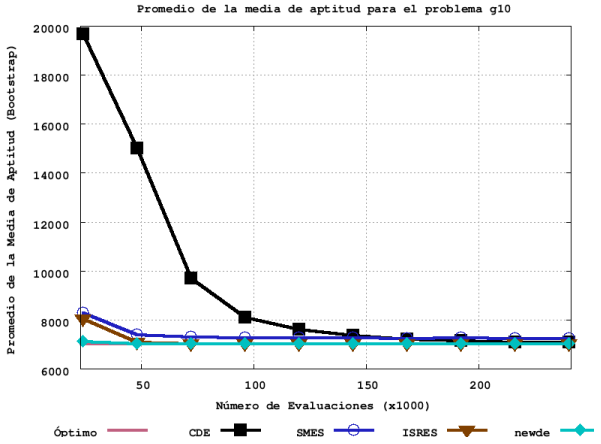
Figura 7.15: Comportamiento de la Aptitud Media conforme se incrementa el número de evaluaciones de la función objetivo para los problemas $g02$, $g03$, $g04$, $g05$, $g06$, $g07$



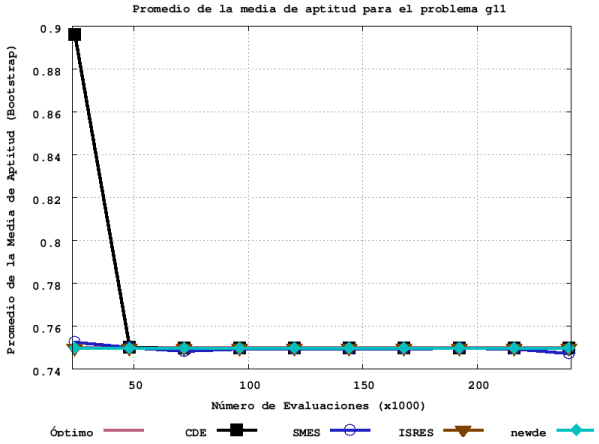
(a) $g08$



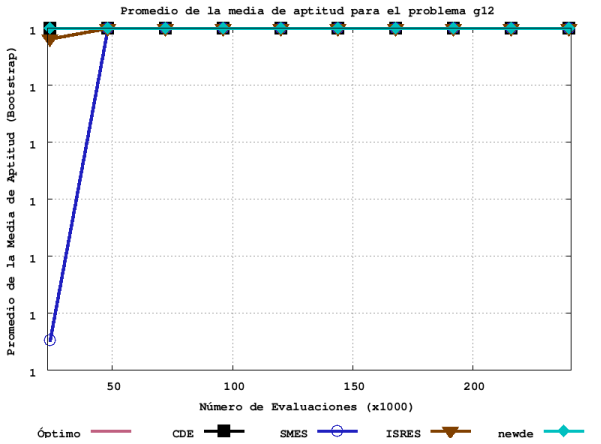
(b) $g09$



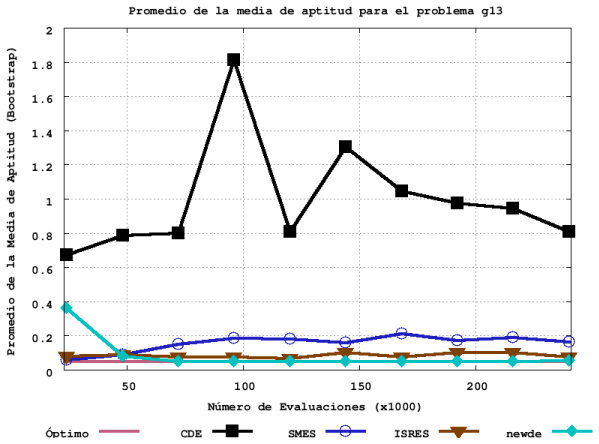
(c) $g10$



(d) $g11$



(e) $g12$



(f) $g13$

Figura 7.16: Comportamiento de la Aptitud Media conforme se incrementa el número de evaluaciones de la función objetivo para los problemas $g08$, $g09$, $g10$, $g11$, $g12$, $g13$

1 **for** $i \leftarrow \{1, \ldots, K\}$ **do**
2 Generar un re-muestra con reemplazo $M_i^* = \{y_1, y_2, \ldots, y_n\}$ de la muestra original $M = x_1, x_2, \ldots, x_n$, donde la probabilidad de cada elemento x_i es $\frac{1}{n}$.
3 Calcular el valor $\hat{d}_p(M_i^*)$ de la estadística $\hat{d}_p$ para la re-muestra M_i^* en el problema p.
4 Los valores $\hat{d}_p(M_1^*), \ldots, \hat{d}_p(M_K^*)$ componen a la **distribución muestral de la distancia promedio al valor óptimo**.

Algoritmo 14: Procedimiento de Bootstrap para calcular la distribución de probabilidades muestral para la estadística $\hat{d}_p$

A partir de la distribución muestral obtenida por el procedimiento de Bootstrap (algoritmo 14) se calcula el valor $\hat{\epsilon}_{95\,\%}$ tal que $Pr\{X \leq \hat{\epsilon}_{95\,\%}\} = 0.95$; esto es, con una probabilidad del 95 % el intervalo $x_p^* \pm \hat{\epsilon}_{95\,\%}$ contiene a las soluciones resultado de ejecuciones sucesivas independientes del algoritmo en cuestión.

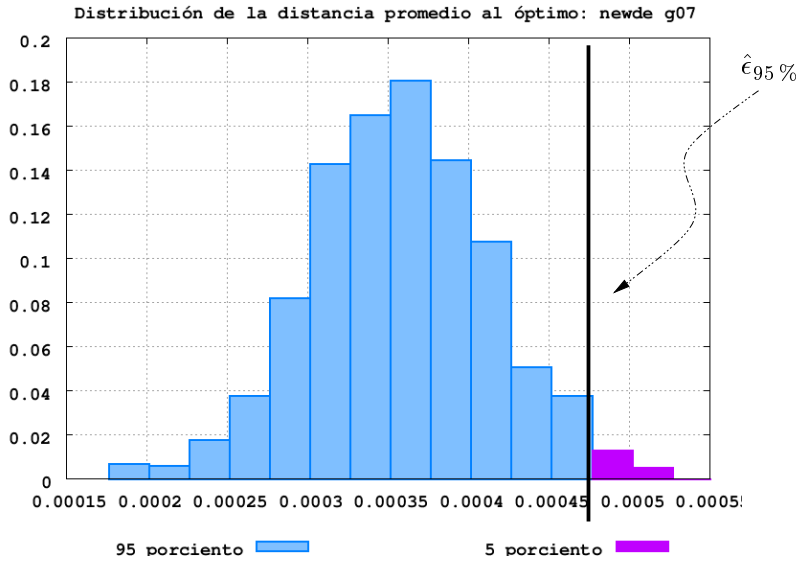


Figura 7.17: Distribución Muestral de Probabilidades para el algoritmo **newde**, problema $g07$

En la figura 7.17 se muestra la distribución muestral de la distancia promedio al óptimo para el algoritmo **newde** en el problema $g07$. Se ilustra también el cálculo del valor $\hat{\epsilon}_{95\,\%}$.

En el Apéndice C.2 se muestran las distribuciones muestrales para el algoritmo **newde** en todos los problemas; en la mayoría de estas gráficas se observa un distribución que se ajusta a una forma normal. En los problemas $g05$, $g06$, $g11$ y $g12$ se no se observa una distribución dado que el algoritmo generó una única solución: el valor óptimo.

Empleando el procedimiento anterior se calcularon los valores de $\hat{\epsilon}_{95\,\%}$ para cada par **algoritmo-problema** utilizando $240,000$ evaluaciones de la función objetivo. Los resultados se muestran en la tabla 7.12, donde los valores marcados en **negritas*** representan los mejores valores para cada problema. De la tabla se puede observar que el algoritmo **newde** exhibe los mejores resultados en todos los problemas a excepción de $g01$. En este último caso el valor de $\hat{\epsilon}_{95\,\%}$ es muy pequeño. Se observa además que el algoritmo **CDE** muestra resultados muy pobres en los problemas $g01$, $g05$, $g10$ y $g13$.

problema	cde	smes	isres	newde
$g01$	1.09	$\mathbf{0.00}^*$	9.05×10^{-12}	1.82×10^{-12}
$g02$	3.96×10^{-1}	3.02×10^{-2}	5.48×10^{-2}	$\mathbf{2.78 \times 10^{-3}}^*$
$g03$	7.86×10^{-1}	2.49×10^{-3}	$\mathbf{5.00 \times 10^{-4}}^*$	$\mathbf{5.00 \times 10^{-4}}^*$
$g04$	$\mathbf{3.28 \times 10^{-4}}^*$	$\mathbf{3.28 \times 10^{-4}}^*$	$\mathbf{3.28 \times 10^{-4}}^*$	$\mathbf{3.28 \times 10^{-4}}^*$
$g05$	$2.83 \times 10^{+2}$	$1.13 \times 10^{+2}$	$\mathbf{1.39 \times 10^{-3}}^*$	$\mathbf{1.39 \times 10^{-3}}^*$
$g06$	$\mathbf{4.42 \times 10^{-6}}^*$	$2.72 \times 10^{+1}$	$\mathbf{4.42 \times 10^{-6}}^*$	$\mathbf{4.42 \times 10^{-6}}^*$
$g07$	5.06×10^{-1}	2.52×10^{-1}	5.67×10^{-4}	$\mathbf{4.01 \times 10^{-4}}^*$
$g08$	$\mathbf{4.14 \times 10^{-8}}^*$	9.38×10^{-7}	$\mathbf{4.14 \times 10^{-8}}^*$	$\mathbf{4.14 \times 10^{-8}}^*$
$g09$	1.42×10^{-7}	2.95×10^{-2}	7.50×10^{-8}	$\mathbf{7.49 \times 10^{-8}}^*$
$g10$	$4.33 \times 10^{+1}$	$2.53 \times 10^{+2}$	8.26×10^{-2}	$\mathbf{6.92 \times 10^{-2}}^*$
$g11$	$\mathbf{1.00 \times 10^{-4}}^*$	2.04×10^{-2}	$\mathbf{1.00 \times 10^{-4}}^*$	$\mathbf{1.00 \times 10^{-4}}^*$
$g12$	$\mathbf{0.00}^*$	$\mathbf{0.00}^*$	$\mathbf{0.00}^*$	$\mathbf{0.00}^*$
$g13$	8.63×10^{-1}	2.34×10^{-1}	1.12×10^{-1}	$\mathbf{8.29 \times 10^{-6}}^*$

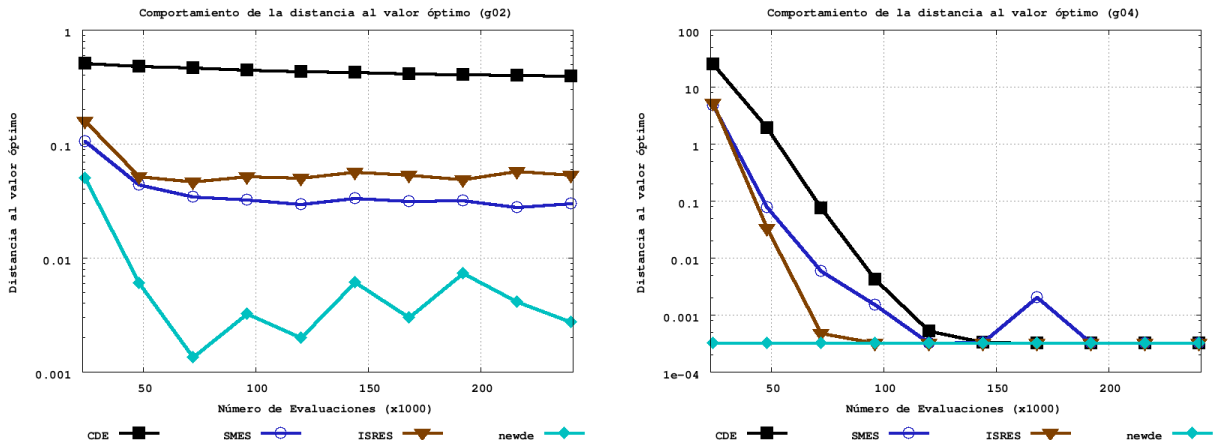
Tabla 7.12: Valores de $\hat{\epsilon}_{95\,\%}$ para cada par **problema-modelo**. Los mejores valores para cada problema están marcados en **negritas***

7.2.5. Gráficas de Crecimiento de $\hat{\epsilon}_{95\,\%}$

Con la finalidad de apreciar el comportamiento del valor de $\hat{\epsilon}_{95\,\%}$ para cada uno de los algoritmos a lo largo del proceso de búsqueda, se generó un conjunto de 100 ejecuciones independientes por cada par **algoritmo-problema** para diferente número de evaluaciones tal como se mostró en la sección 7.2.3.

En la gráfica 7.18 se muestra el comportamiento de $\hat{\epsilon}_{95\,\%}$ conforme se incrementa el número de evaluaciones de la función objetivo para los problemas $g02$ y $g04$. En estas gráfi-

cas se aprecia nuevamente que el algoritmo **newde** obtiene los mejores resultados a lo largo de las diferentes cantidades de evaluaciones de la función objetivo. El algoritmo es capaz de encontrar el vecindario del valor óptimo en muy pocas evaluaciones. Se observa además que de los cuatro algoritmos, **CDE** muestra el peor rendimiento. Este mismo comportamiento se aprecia en la mayoría de las gráficas de $\hat{\epsilon}_{95\,\%}$ para los demás problemas de prueba; estas gráficas se muestran en el Apéndice C.3. Nótese que estas gráficas están en escala logarítmica.



(a) Problema $g02$

(b) Problema $g04$

Figura 7.18: Comportamientos del valor $\hat{\epsilon}_{95\,\%}$ para los problemas $g02$ y $g04$

Conforme los algoritmos generan consistentemente soluciones cercanas al óptimo, el valor $\hat{\epsilon}_{95\,\%}$ disminuye no importando si el problema es de minimización o maximización; de hecho, $\hat{\epsilon}_{95\,\%}$ es un valor absoluto. Al normalizar las gráficas de $\hat{\epsilon}_{95\,\%}$ mostradas en el Apéndice C.3 se pierde la información acerca de la *precisión* de cada algoritmo en cada problema pero, se hacen comparables los resultados entre problemas dado que se mantienen las proporciones relativas que guardan los algoritmo entre sí. En la gráfica 7.19 se muestra el comportamiento promedio de $\hat{\epsilon}_{95\,\%}$ a lo largo de todos los problemas de prueba.

Se observa de la gráfica 7.19 una clara diferencia de comportamiento para cada uno de los algoritmos; nótese que la escala de $\hat{\epsilon}_{95\,\%}$ es logarítmica. La nueva propuesta **newde** muestra de manera consistente el mejor rendimiento; ésta obtiene valores de $\hat{\epsilon}_{95\,\%}$ menores que los demás algoritmos a lo largo de todo el proceso de búsqueda. De hecho, en promedio, los algoritmos **CDE** y **SMES** no logran igualar los resultados de **newde** aún empleando

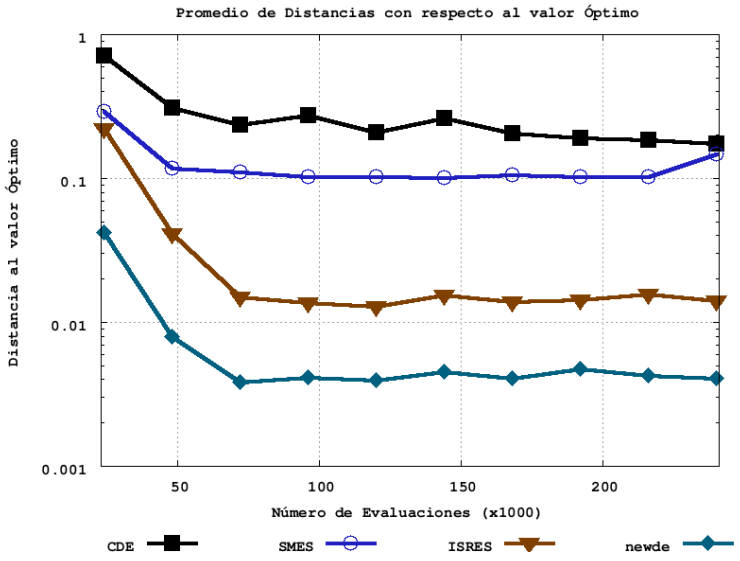


Figura 7.19: Comportamiento promedio de $\hat{\epsilon}_{95\,\%}$ a lo largo de todos los problemas de prueba

diez veces más evaluaciones. Por otro lado, se observa una clara diferencia de comportamiento entre los algoritmos **ISRES** y **newde**. Aún cuando los algoritmos muestran un comportamiento asintótico, la diferencia entre ellos es considerable, recuérdese que la escala es logarítmica. De la gráfica se concluye que el algoritmo **newde** muestra una mayor velocidad de convergencia que los demás algoritmos.

7.2.6. Distribución Muestral de Diferencia de Medias

En las pruebas anteriores se comparan los resultados de los algoritmo de manera indirecta, es decir se hace una estimación de algún parámetro de la población y se comparan los valores de las estimaciones.

Gracias a la gran flexibilidad de los procedimientos de Bootstrap, es posible obtener distribuciones muestrales de parámetros que incluyen a dos o más muestras a la vez. Es decir, se puede obtener la distribución muestral de estadísticas binarias, por ejemplo: la covarianza entre dos muestras, o la diferencia de las medias muestrales.

Se propone la siguiente forma de comparación binaria entre las poblaciones de soluciones de dos algoritmos A y B. En el algoritmo 15 se muestra el procedimiento de Bootstrap para el cálculo de la distribución muestral de la diferencia de medias de población de las muestras

$M_A = \{a_1, \ldots, a_n\}$ y $M_B = \{b_1, \ldots, b_m\}$. A partir de esta distribución es posible calcular la probabilidad de que el comportamiento promedio (media de población) del algoritmo A sea mejor que el del algoritmo B, esto es, $Pr\{\mu_A < \mu_B\}$ o $Pr\{\mu_A > \mu_B\}$ según sea el caso de minimización o maximización respectivamente.

1 **for** $i \leftarrow \{1, \ldots, K\}$ **do**
2 Generar un re-muestra con reemplazo $M^*_{A_i} = \{y_1, \ldots, y_n\}$ de la muestra $M_A = \{a_1, \ldots, a_n\}$ con igual probabilidad y calcular la media muestral $\overline{M_{A_i}}$.
3 Generar un re-muestra con reemplazo $M^*_{B_i} = \{x_1, \ldots, x_m\}$ de la muestra $M_B = \{b_1, \ldots, b_m\}$ con igual probabilidad y calcular la media muestral $\overline{M_{B_i}}$
4 Calcular el valor $\mu^*_{diff_i} = \overline{M_{A_i}} - \overline{M_{B_i}}$
5 Los valores $\{\mu^*_{diff_1}, \ldots, \mu^*_{diff_K}\}$ componen a la **distribución muestral de la diferencia de medias de población**.

Algoritmo 15: Procedimiento de Bootstrap para calcular la distribución muestral de la diferencia de medias de dos algoritmos

En general $Pr\{\mu_A \prec \mu_B\} + Pr\{\mu_B \prec \mu_A\} \neq 1$, puesto que es posible que $Pr\{\mu_A = \mu_B\} \neq 0$ ya que la distribución de probabilidades es discreta; estrictamente hablando se tiene que $Pr\{|\mu_A - \mu_B| \leq \epsilon\} > 0$ para un valor de ϵ dependiente del número de remuestreos en el procedimiento de Bootstrap. En este caso ($\prec$) representa a ($<$) en caso de minimización o ($>$) para maximización. En la tabla 7.13 se muestra la comparación de los algoritmos **newde** y **ISRES** en todas las funciones de prueba empleando $48,000$ evaluaciones de la función objetivo. De la tabla se observa que el promedio de la probabilidad $Pr\{\mu_{newde} \prec \mu_{isres}\}$ es mucho mayor que $Pr\{\mu_{isres} \prec \mu_{newde}\}$.

algo	$g01$	$g02$	$g03$	$g04$	$g05$	$g06$	$g07$	$g08$	$g09$	$g10$	$g11$	$g12$	$g13$	**prom.**
$Pr\{\mu_{newde} \prec \mu_{isres}\}$	**1**	**1**	0	**1**	**1**	**1**	**1**	0	**1**	**1**	**1**	0	0.712	0.747
$Pr\{\mu_{isres} \prec \mu_{newde}\}$	0	0	**1**	0	0	0	0	0	0	0	0	0	0.278	0.0983

Tabla 7.13: Probabilidades $Pr\{\mu_{newde} \prec \mu_{isres}\}$ y $Pr\{\mu_{isres} \prec \mu_{newde}\}$ para cada uno de los poblemas de prueba, empleando $48,000$ evaluaciones

Procediendo de la misma manera que en las secciones 7.2.3 y 7.2.5 se calcula el comportamiento promedio de $Pr\{\mu_{newde} \prec \mu_B\}$ y $Pr\{\mu_B \prec \mu_{newde}\}$ donde $B = \{cde, smes, isres\}$ para diferentes cantidades de evaluaciones de la función objetivo. En la gráfica 7.20 se muestran las gráficas de comportamiento. Existe una diferencia notable entre **newde** y los algoritmos **CDE** y **SMES**, tal diferencia se mantiene casi constante para todos los valores de evaluaciones empleados. Por otro lado, comparando **newde** y **ISRES** se observa

un decrecimiento de $Pr\{\mu_{newde} \prec \mu_{isres}\}$ conforme aumenta el número de evaluaciones. Sin embargo, esto se debe a que $Pr\{\mu_{newde} = \mu_{isres}\}$ aumenta ya que los dos algoritmos encuentran consistentemente el valor óptimo de varios problemas y no corresponde a un crecimiento de $Pr\{\mu_{isres} \prec \mu_{newde}\}$. En todo momento $Pr\{\mu_{newde} < \mu_{isres}\}$ es mayor que $Pr\{\mu_{isres} \prec \mu_{newde}\}$.

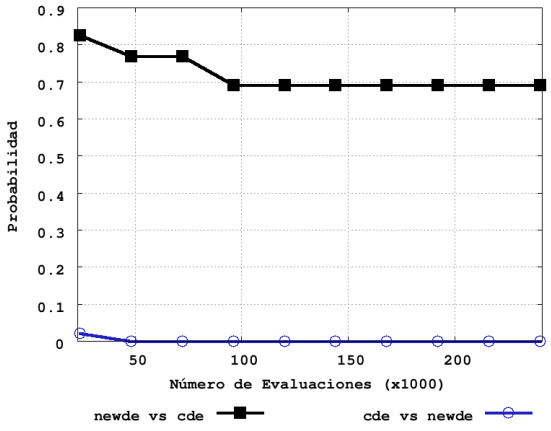
En base a los resultados de las diferentes pruebas se concluye que el algoritmo **newde** presenta el mejor rendimiento. A continuación se listan los algoritmos de acuerdo a su rendimiento obtenido.

1. Nuevo modelo de Evolución Diferencial para espacios restringidos **(newde)**
2. Versión mejorada de las Jerarquías Estocásticas **(ISRES)** [81]
3. Estrategia Evolutiva Multimiembro Simple **(SMES)** [60]
4. Evolución Diferencial Restringida **(CDE)** [55]

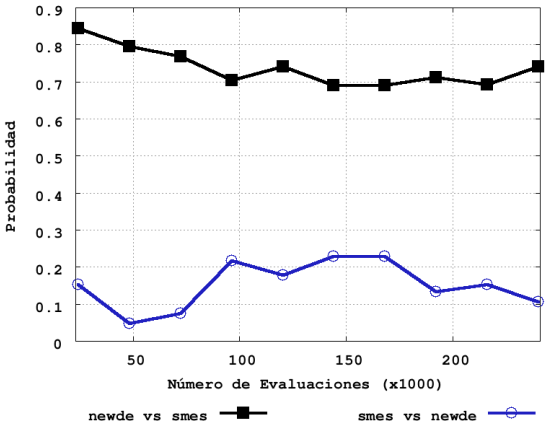
7.3. Resumen del Capítulo

En este capítulo se presentaron los resultados experimentales del algoritmo propuesto de Evolución Diferencial. Se concluyó que el nuevo modelo **newde** mejora de manera significativa el rendimiento del algoritmo original (modelo **rand/1/bin**) así como a otras propuestas presentadas en la literatura. Combinar información global (mejor solución en la población) con información local (mejor solución en la trayectoria de un individuo) probó mejorar sustancialmente la velocidad de convergencia del algoritmo, sin sacrificar su capacidad de exploración.

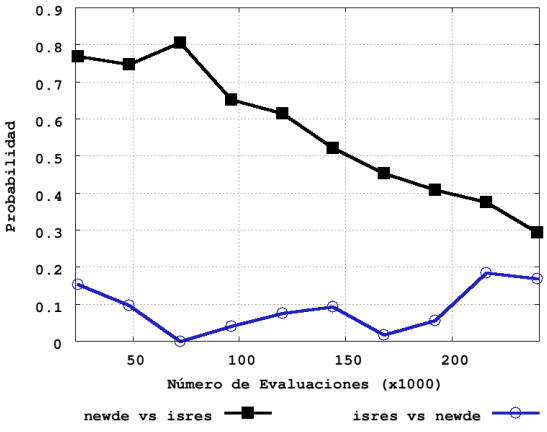
Por otro lado, el mecanismo de manejo de restricciones empleado proporciona elementos que hacen al algoritmo propuesto una técnica competitiva para optimización númerica con restricciones. A lo largo de diferentes pruebas, la propuesta demostró un mejor rendimiento tanto en calidad de soluciones como en consistencia de las mismas al ser comparada contra algoritmos representativos del área de computación evolutiva para optimización con restricciones.



(a) μ_{newde} vs μ_{cde}



(b) μ_{newde} vs μ_{smes}



(c) μ_{newde} vs μ_{isres}

Figura 7.20: Gráficas de comportamiento promedio de $Pr\{\mu_{newde} \prec \mu_B\}$ y $Pr\{\mu_B \prec \mu_{newde}\}$ conforme aumenta el número de evaluaciones de la función objetivo, para $B = \{cde, smes, isres\}$

Capítulo 8

Conclusiones y Trabajo Futuro

La Computación Evolutiva empleada en optimización numérica con restricciones es un área activa de investigación hoy en día. A pesar de que se han desarrollado diversos algoritmos competitivos aún se siguen explorando nuevas posibilidades. De entre estos nuevos desarrollos, en esta tesis se exploran las posibilidades del Algoritmo de Evolución Diferencial como optimizador de espacios restringidos.

El algoritmo original de Evolución Diferencial ha mostrado ser un método *robusto* de búsqueda en una amplia gama de problemas [75]. Por lo tanto, resulta atractivo emplearlo en la optimización de espacios restringidos. En la literatura de la Evolución Diferencial [75, 54] se han utilizado muy ampliamente las funciones de penalización como técnica de manejo de restricciones. Sin embargo, éstas presentan una serie de desventajas siendo quizás la más importante la del ajuste que requieren los coeficientes de penalización, ya que éstos son dependientes del problema [6].

De tal manera, en esta tesis se presenta un mecanismo de manejo de restricciones basado en funciones de penalización pero, a diferencia de las otras propuestas, no se requiere ajustar ningún coeficiente de penalización. Esta propuesta difiere de otros intentos por ajustar estos coeficientes dinámicamente o auto-adaptándolos, ya que evita la sobre- y sub- penalización manteniendo un balance entre comparaciones basadas en la función objetivo y el grado de penalización de las restricciones.

Además, se propone un nuevo modelo de Evolución Diferencial que incorpora información global (la mejor solución de la población) junto con información histórica (la mejor solución en la trayectoria de cada individuo en la población). Aunado a esto, se incorpora un buscador local que mejora la capacidad de explotación del algoritmo. Esta propuesta se comparó con otros modelos encontrados en la literatura de ED. Específicamente contra los

modelos: **rand/1/bin** siendo éste el modelo original, más ampliamente usado y con buen rendimiento, **best/1/bin** siendo éste el modelo que presentó el mejor rendimiento en un estudio comparativo empleando trece funciones de prueba estándares en la optimización sin restricciones con algoritmos evolutivos.

De los resultados experimentales se concluye que la propuesta obtiene claramente mejores resultados conforme el tamaño del espacio de búsqueda crece, logrando incluso diferencias de órdenes de magnitud en los resultados.

La propuesta de ED en espacios restringidos se compone del nuevo mecanismo de manejo de restricciones junto al nuevo modelo de evolución diferencial. Ésta se compara contra algoritmos representativos del estado del arte, incluyendo a la mejor técnica de optimización restringida conocida a la fecha (denominada Jerarquías Estocásticas).

Se efectuaron una serie de experimentos con la finalidad de validar el rendimiento de la propuesta, a pesar de tratarse de un algoritmo estocástico. Éstos se validan usando procedimientos estadísticos de cómputo intensivo que nos permiten obtener aproximaciones a distribuciones muestrales de diferentes estadísticas además de la media sin suponer normalidad en las muestras. Se propone además, emplear al valor $\hat{\epsilon}_{95\,\%}$ como estimador de dispersión de las soluciones con respecto al valor óptimo global. Así también se propone un mecanismo de comparación binaria entre algoritmos para determinar la probabilidad de que un algoritmo obtenga un comportamiento promedio mejor que otro.

En todas las pruebas se mostró consistentemente que esta propuesta obtiene mejores resultados, inclusive superando a los de la versión mejorada de la Jerarquías Estocásticas, requiriendo además menor número de evaluaciones de la función objetivo. Es importante mencionar que el algoritmo propuesto de ED no sólo se presenta como un algoritmo competitivo en la optimización con restricciones sino que además se presenta un nuevo modelo que puede ser empleado en la optimización de funciones sin restricciones.

Por otro lado, resulta interesante notar que tanto la Estrategia Evolutiva Multimiembro Simple (SMES) así como la versión mejorada de las Jerarquías Estocásticas (ISMES) emplean un operador de recombinación compuesto de una diferenciación junto con una cruza discreta. Tal recombinación caracteriza a la ED. Es interesante notar que los autores de ISRES [81] y SMES [80] mencionan que la calidad de los resultados se incrementa al emplear diferenciaciones. De hecho, la propuesta de esta tesis, la Estrategia Evolutiva Multimiembro Simple y la versión mejorada de las Jerarquías Estocásticas (ISMES) comparten elementos en común: el mismo tipo de representación, la generación de múltiples descendientes, emplean una recombinación estilo Evolución Diferencial y los mecanismos de manejo de restricciones buscan un balance entre individuos factibles e infactibles.

A pesar de que cada uno de ellos propone estos elementos de manera independiente y con diferentes justificaciones, se abre la posibilidad de una probable unificación de estos conceptos en un nuevo algoritmo que no forzosamente puede encajar en la descripción de una Estrategia Evolutiva o de la Evolución Diferencial. Esta es una línea abierta de investigación.

Por otro lado, aún no se han explorado las posibilidades de auto-adaptación de los parámetros de control de la Evolución Diferencial. Resulta particularmente atractiva la auto-adaptación del parámetro CR que controla la cruza discreta y que repercute directamente en el rendimiento del algoritmo. Esto abre la posibilidad de una ED con parámetros mínimos.

Finalmente, el modelo propuesto de ED muestra cierta similitud con la función de vuelo empleada en la optimización con cúmulos de partículas [48] y han habido intentos por combinar ambas propuestas, pero hasta ahora con resultados marginales. Ésta también es un área abierta a la investigación.

Apéndice A

Funciones de prueba sin restricciones

$f01$ - Modelo de esfera

$$f_1(x) = \sum_{i=1}^{30} x_i^2$$

$$-100 \leq x_i \leq 100$$
$$min(f_1) = f_1(0, ..., 0) = 0$$

$f02$ - Problema de Schwefel 2.22

$$f_2(x) = \sum_{i=1}^{30} |x_i| + \prod_{i=1}^{30} |x_i|$$

$$-10 \leq x_i \leq 10$$
$$min(f_2) = f_2(0, ..., 0) = 0$$

$f03$ - Problema de Schwefel 1.2

$$f_3(x) = \sum_{i=1}^{30} \left(\sum_{j=1}^{i} x_j \right)^2$$

$$-100 \leq x_i \leq 100$$
$$min(f_3) = f_3(0, ..., 0) = 0$$

$f04$ - Problema de Schwefel 2.21

$$f_4(x) = max_i\{|x_i|, 1 \leq i \leq 30\}$$

$$-100 \leq x_i \leq 100$$
$$min(f_4) = f_4(0, ..., 0) = 0$$

$f05$ - Función generalizada de Rosenbrock

$$f_5(x) = \sum_{i=1}^{29} |100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2|$$

$$-30 \leq x_i \leq 30$$
$$min(f_5) = f_5(1, ..., 1) = 0$$

$f06$ - Función de paso

$$f_6(x) = \sum_{i=1}^{30} (\lfloor x_i + 0.5 \rfloor)^2$$

$$-100 \leq x_i \leq 100$$
$$min(f_6) = f_6(0, ..., 0) = 0$$

$f07$ - Función cuadrática con ruido

$$f_7(x) = \sum_{i=1}^{30} ix_i^4 + random[0, 1)$$

$$-1.28 \leq x_i \leq 1.28$$
$$min(f_7) = f_7(0, ..., 0) = 0$$

$f08$ - Problema generalizado de Schwefel 2.26

$$f_8(x) = \sum_{i=1}^{30} \left(x_i sin(\sqrt{|x_i|}) \right)$$

$$-500 \leq x_i \leq 500$$
$$min(f_8) = f_8(420.9687, ..., 420.9687) = -12569.5$$

$f09$ - Función generalizada de Rastrigin

$$f_9(x) = \sum_{i=1}^{30}[x_i^2 - 10cos(2\pi x_i) + 10]$$

$$-5.12 \leq x_i \leq 5.12$$
$$min(f_9) = f_9(0, ..., 0) = 0$$

$f10$ - Función de Ackley

$$f_{10}(x) = -20exp\left(-0.2\sqrt{\frac{1}{30}\sum_{i=1}^{30}x_i^2)}\right) - exp\left(\frac{1}{30}\sum_{i=1}^{30}cos(2\pi x_i)\right) + 20 + e$$

$$-32 \leq x_i \leq 32$$
$$min(f_{10}) = f_{10}(0, ..., 0) = 0$$

$f11$ - Función generalizada de Griewank

$$f_{11}(x) = \frac{1}{4000}\sum_{i=1}^{30}x_i^2 - \prod_{i=1}^{30}cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$$

$$-600 \leq x_i \leq 600$$
$$min(f_{11}) = f_{11}(0, ..., 0) = 0$$

$f12, f13$ - Funciones generalizadas y penalizadas

$$f_{12}(x) = \frac{\pi}{30}\left\{10sin^2(\pi y_1) + \sum_{i=1}^{29}(y_i - 1)^2[1 + 10sin^2(\pi y_{i+1})] + (y_n - 1)^2\right\} +$$

$$\sum_{i=1}^{30}u(x_i, 10, 100, 4)$$

$$-50 \leq x_i \leq 50$$
$$min(f_{12}) = f_{12}(1, ..., 1) = 0$$

$$f_{13}(x) = 0.1\left\{sin^2(\pi 3x_1) + \sum_{i=1}^{29}(x_i - 1)^2[1 + sin^2(3\pi x_{i+1})] + (x_n - 1)^2[1 + sin^2(2\pi x_{30})]\right\} +$$

$$\sum_{i=1}^{30} u(x_i, 5, 100, 4)$$

$$-50 \leq x_i \leq 50$$
$$min(f_{13}) = f_{13}(1, ..., 1) = 0$$

donde:

$$u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & x_i > a \\ 0, & -a \leq x_i \leq a \\ k(-x_i - a)^m, & x_i < -a \end{cases}$$

$$y_i = 1 + \frac{1}{4}(x_i + 1)$$

Apéndice B

Funciones de prueba con restricciones

g01
Minimizar:

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

sujeto a:

$$\begin{aligned} g(x)_1 &= 2x_1 + 2x_2 + x_{10} + x_{11} - 10 \leq 0 \\ g(x)_2 &= 2x_1 + 2x_3 + x_{10} + x_{12} - 10 \leq 0 \\ g(x)_3 &= 2x_2 + 2x_3 + x_{11} + x_{12} - 10 \leq 0 \\ g(x)_4 &= -8x_1 + x_{10} \leq 0 \\ g(x)_5 &= -8x_2 + x_{11} \leq 0 \\ g(x)_6 &= -8x_3 + x_{12} \leq 0 \\ g(x)_7 &= -2x_4 - x_5 + x_{10} \leq 0 \\ g(x)_8 &= -2x_6 - x_7 + x_{11} \leq 0 \\ g(x)_9 &= -2x_8 - x_9 + x_{12} \leq 0 \end{aligned}$$

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

$min(f) = f(1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 1) = -15$

g02
Maximizar:

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

sujeto a:

$$g(x)_1 = 0.75 - \prod_{i=1}^{n} x_i \leq 0$$
$$g(x)_1 = \sum_{i=1}^{n} x_i - 7.5n \leq 0$$

$n = 20,\ 0 \leq x_i \leq 10 (i = 1, ..., n)$

El óptimo es deconocido. La mejor aproximación encontrada es:

$max(f) = f(x*) = -0.803619$

g03

Maximizar:

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

sujeto a:

$$h(x)_1 = \sum_{i=1}^{n} x_i^2 - 1 = 0$$

$n = 0,\ 0 \leq x_i \leq 1$

$x^* = \frac{1}{\sqrt{n}} (i = 1, ..., n),\ max(f) = f(x^*) = 1$

g04

Minimizar:

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

sujeto a:

$$g(x)_1 = 85.334407 + 0.0056858x_2x_5 + 0.0006262x_1x_4 - 0.0022053x_3x_5 - 92.0 \leq 0$$
$$g(x)_2 = -85.334407 - 0.0056858x_2x_5 - 0.0006262x_1x_4 + 0.0022053x_3x_5 \leq 0$$
$$g(x)_3 = 80.51249 + 0.0071317x_2x_5 + 0.0029955x_1x_2 + 0.0021813x_3^2 - 110.0 \leq 0$$

$$g(x)_4 = 80.51249 - 0.0071317x_2x_5 - 0.0029955x_1x_2 - 0.0021813x_3^2 + 90.0 \leq 0$$
$$g(x)_5 = 9.300961 + 0.0047026x_3x_5 + 0.0012547x_1x_3 + 0.0019085x_3x_4 - 25.0 \leq 0$$
$$g(x)_6 = 9.300961 - 0.0047026x_3x_5 - 0.0012547x_1x_3 - 0.0019085x_3x_4 + 20.0 \leq 0$$

$78 \leq x_1 \leq 122$, $33 \leq x_2 \leq 45$ y $27 \leq x_i \leq 45 (i = 3, 4, 5)$

$min(f) = f(78, 33, 29.995256025682, 45, 36.775812905788) = -30665.539$

g05

Minimizar:

$$f(x) = 3x_1 + 0.000001x_i^3 + 2x_2 + (0.000002/3)x_2^3$$

sujeto a:

$$g(x)_1 = -x_4 + x_3 - 0.55 \leq 0$$
$$g(x)_2 = -x_3 + x_4 - 0.55 \leq 0$$
$$h(x)_3 = 1000sin(-x_3 - 0.25) + 1000sin(-x_4 - 0.25) + 894.8 - x_1 = 0$$
$$h(x)_4 = 1000sin(x_3 - 0.25) + 1000sin(x_3 - x_4 - 0.25) + 894.8 - x_2 = 0$$
$$h(x)_5 = 1000sin(x_4 - 0.25) + 1000sin(x_4 - x_3 + 0.25) + 1294.8 = 0$$

$0 \leq x_i \leq 1200 (i = 1, 2)$ y $-0.55 \leq x_i \leq 0.55 (i = 3, 4)$

$min(f) = f(679.9453, 1026.067, 0.1188764, -0.3962336) = 5126.4981$

g06

Minimizar:

$$f(x) = (x_1 - 10)^3 + (x_2 - 20)^3$$

sujeto a:

$$g(x)_1 = -(x_1 - 5)^2 - (x_2 - 5)^2 + 100 \leq 0$$
$$g(x)_2 = (x_1 - 6)^2 + (x_2 - 5)^2 - 86.81 \leq 0$$

$13 \leq x_1 \leq 100$, $0 \leq x_2 \leq 100$

$min(f) = f(14.095, 0.84296) = -6961.81388$

g07

Minimizar:

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

sujeto a:

$$g(x)_1 = -105+4x_1+5x_2-3x_7+9x_8 \leq 0$$
$$g(x)_2 = 10x_1-8x_2-17x_7+2x_8 \leq 0$$
$$g(x)_3 = -8x_1+2x_2+5x_9-2x_{10}-12 \leq 0$$
$$g(x)_4 = 3(x_1-2)^2+4(x_2-3)^2+2x_3^2-7x_4-120 \leq 0$$
$$g(x)_5 = 5x_1^2+8x_2+(x_3-6)^2-2x_4-40 \leq 0$$
$$g(x)_6 = x_1^2+2(x_2-2)^2-2x_1x_2+14x_5-6x_6 \leq 0$$
$$g(x)_7 = 0.5(x_1-8)^2+2(x_2-4)^2+3x_5^2-x_6-30 \leq 0$$
$$g(x)_8 = -3x_1+6x_2+12(x_9-8)^2-7x_{10} \leq 0$$

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

$min(f) = f(2.171996, 2.363683, 8.773926, 5.095984, 0.9906548,$
$1.430574, 1.321644, 9.828726, 8.280092, 8.375927) = 24.3062091$

g08

Minimizar:

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

sujeto a:

$$g(x)_1 = x_1^2-x_2+1 \leq 0$$
$$g(x)_2 = 1-x_1+(x_2-4)^2 \leq 0$$

$0 \leq x_i \leq 10(i = 1, 2)$

$min(f) = f(1.2279713, 4.2453733) = 0.095825$

g09

Minimizar:

$f(x) = (x_1 - 10)^2 + 5(x_2 - 12)^2 + x_3^2 + 3(x_4 - 11)^2 + 10x_5^6 + 7x_6^2 + x_7^4 - 4x_4x_7 - 10x_6 - 8x_7$

sujeto a:

$$g(x)_1 = -127 + 2x_1^2 + 3x_2^4 + x_3 + 4x_4^2 + 5x_5 \leq 0$$
$$g(x)_2 = -282 + 7x_1 + 3x_2 + 10x_3^2 + x_4x_5 \leq 0$$
$$g(x)_3 = -196 + 23x_1 + x_2^2 + 6x_6^2 - 8x_7 \leq 0$$
$$g(x)_4 = 4x_1^2 + x_2^2 - 3x_1x_2 + 2x_3^2 + 5x_6 - 11x_7 \leq 0$$

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

$min(f) = f(2.330499, 1.951372, -0.4775414, 4.365726, -0.6244870, 1.038131, 1.594227)$ $= 680.6300573$

g10

Minimizar:

$$f(x) = x_1 + x_2 + x_3$$

sujeto a:

$$g(x)_1 = -1 + 0.0025(x_4 + x_6) \leq 0$$
$$g(x)_2 = -1 + 0.0025(x_5 + x_7 - x_4) \leq 0$$
$$g(x)_3 = -1 + 0.01(x_8 - x_5) \leq 0$$
$$g(x)_4 = -x_1x_6 + 833.33252x_4 + 100x_1 - 83333.33 \leq 0$$
$$g(x)_5 = -x_2x_7 + 1250x_5 + x_2x_4 - 1250x_4 \leq 0$$
$$g(x)_6 = -x_3x_8 + 1250000 + x_3x_5 - 2500x_5 \leq 0$$

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

$min(f) = f(579.3167, 1359.943, 5110.071, 182.0174, 295.5985, 217.9799, 286.4162, 395.5979) =$ 7049.3307

g11

Minimizar:

$$f(x) = x_1^2 + (x_2 - 1)^2$$

sujeto a:

$$h(x)_1 = x_2 - x_1^2 = 0$$

$-1 \leq x_1 \leq 1 (i = 1, 2)$

$min(f) = f(\pm\frac{1}{\sqrt{2}}, \frac{1}{2}) = 0.75$

g12

Minimizar:

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

sujeto a:

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

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

$min(f) = f(5, 5, 5) = 1$

g13

Minimizar:

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

sujeto a:

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

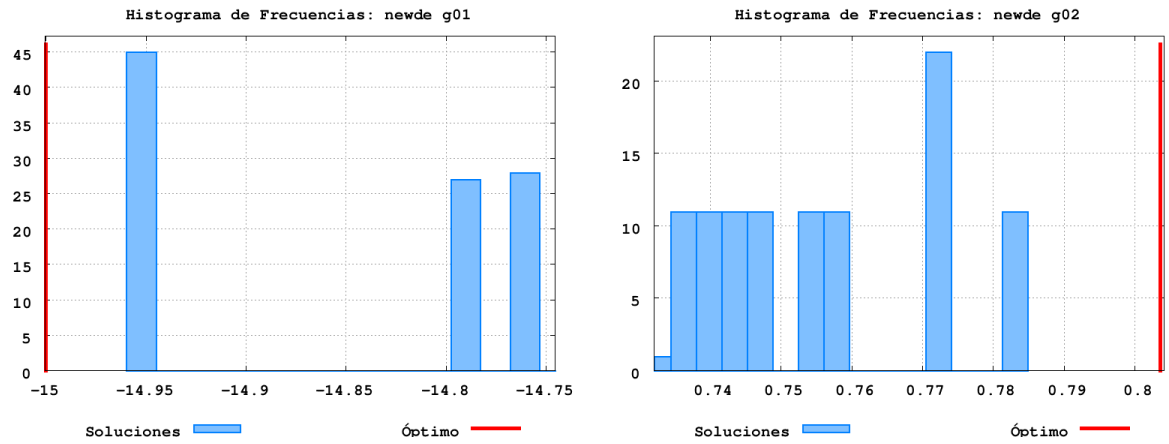
$-2.3 \leq x_i \leq 2.3 (i = 1, 2)$ y $-3.2 \leq x_i \leq 3.2 (i = 3, 4, 5)$

$min(f) = f(-1.717143, 1.595709, 1.827247, -0.7636413, -0.763645) = 0.0539498$

Apéndice C

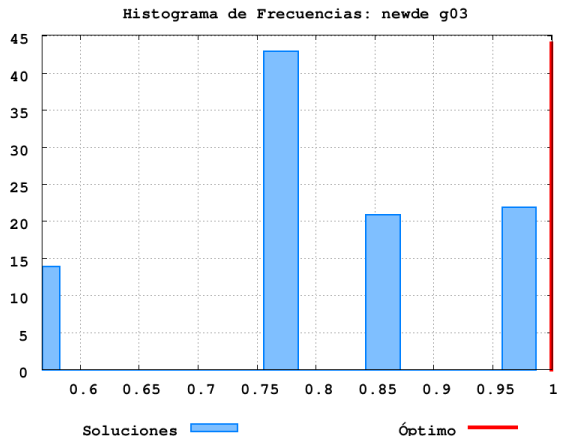
Gráficas de Resultados

C.1. Histogramas de Frecuencia de Soluciones

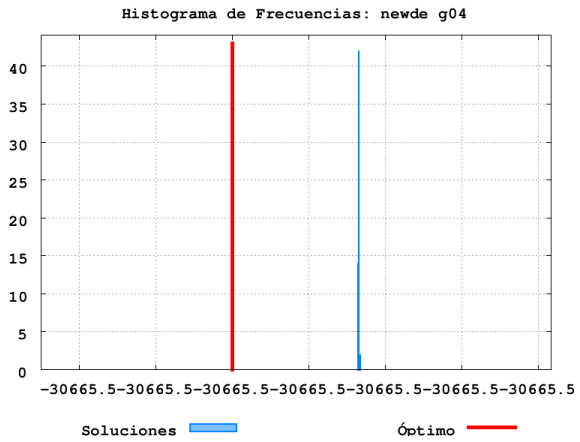


(a) $g01$ (b) $g02$

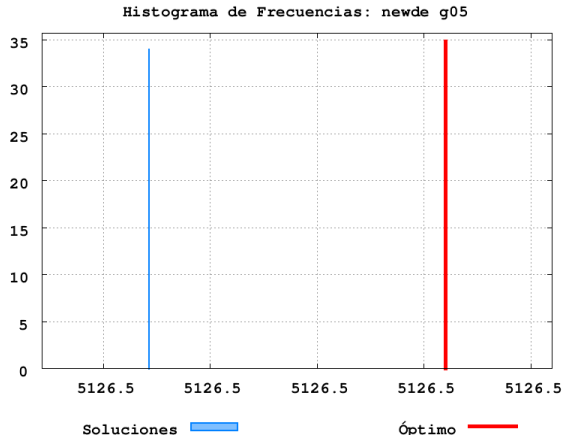
Figura C.1: Histogramas de Frecuencias de las soluciones del algoritmo propuesto **newde** para los problemas de prueba $g01$, $g02$. Se observa de los histogramas que las muestras de 100 ejecuciones independientes no se ajustan a una forma normal.



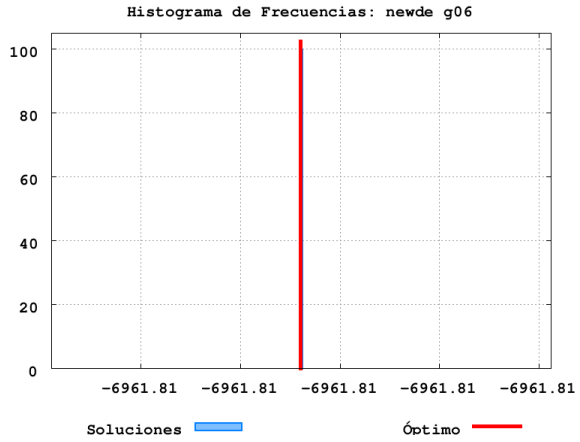
(a) $g03$



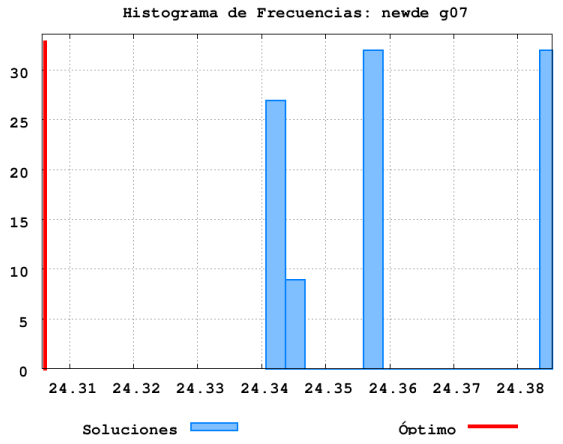
(b) $g04$



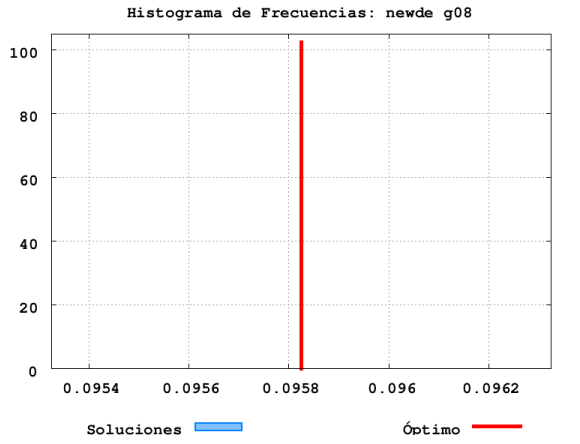
(c) $g05$



(d) $g06$

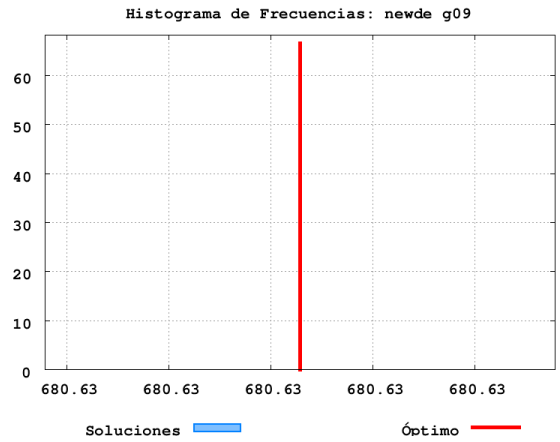


(e) $g07$

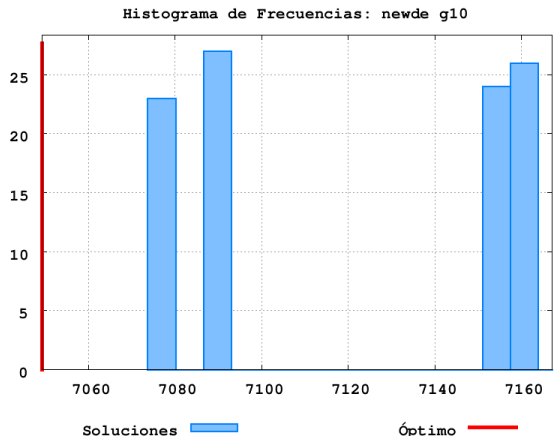


(f) $g08$

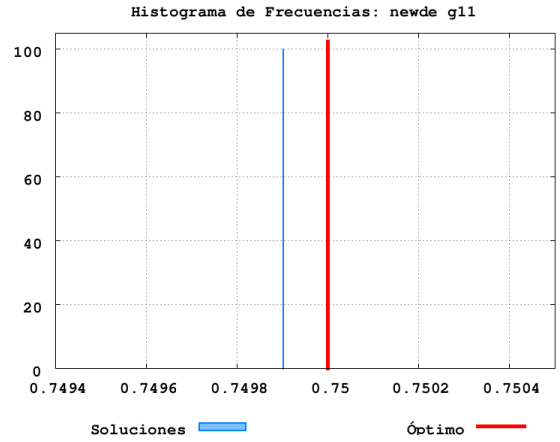
Figura C.2: Histogramas de Frecuencias de las soluciones del algoritmo propuesto **newde** para los problemas de prueba $g03$, $g04$, $g05$, $g06$, $g07$, $g08$. Se observa de los histogramas que las muestras de 100 ejecuciones independientes no se ajustan a una forma normal.



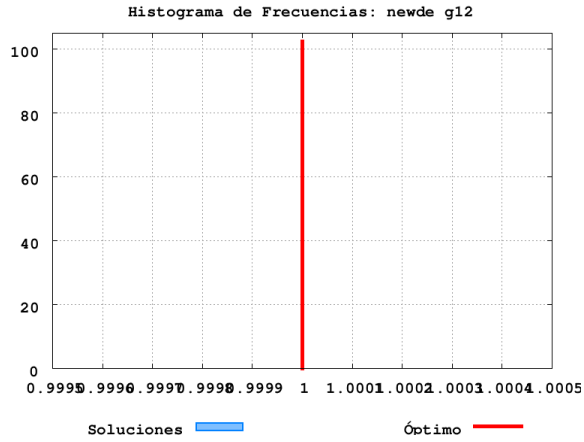
(a) $g09$



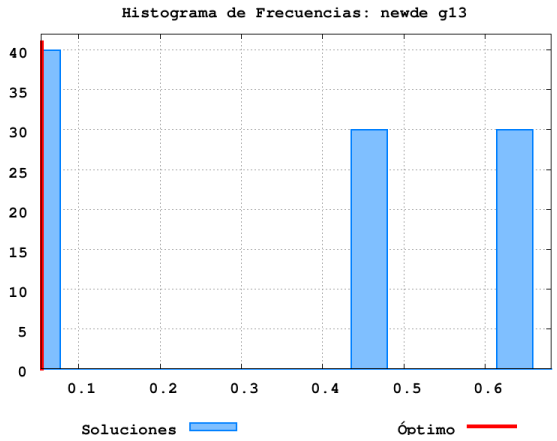
(b) $g10$



(c) $g11$



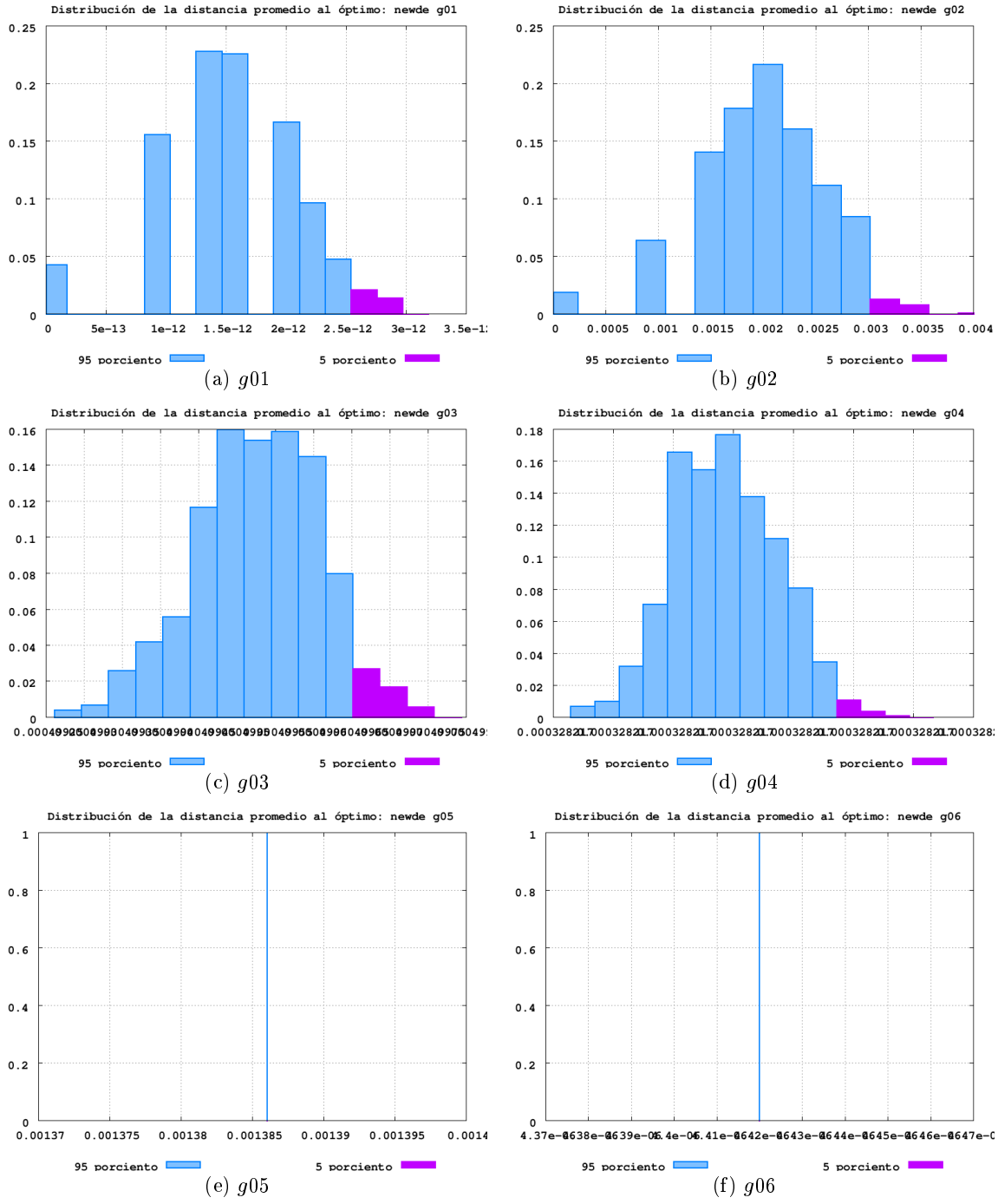
(d) $g12$



(e) $g13$

Figura C.3: Histogramas de Frecuencias de las soluciones del algoritmo propuesto **newde** para los problemas de prueba $g09$, $g10$, $g11$, $g12$, $g13$. Se observa de los histogramas que las muestras de 100 ejecuciones independientes no se ajustan a una forma normal.

C.2. Distribuciones de la Distancia Promedio al Valor Óptimo

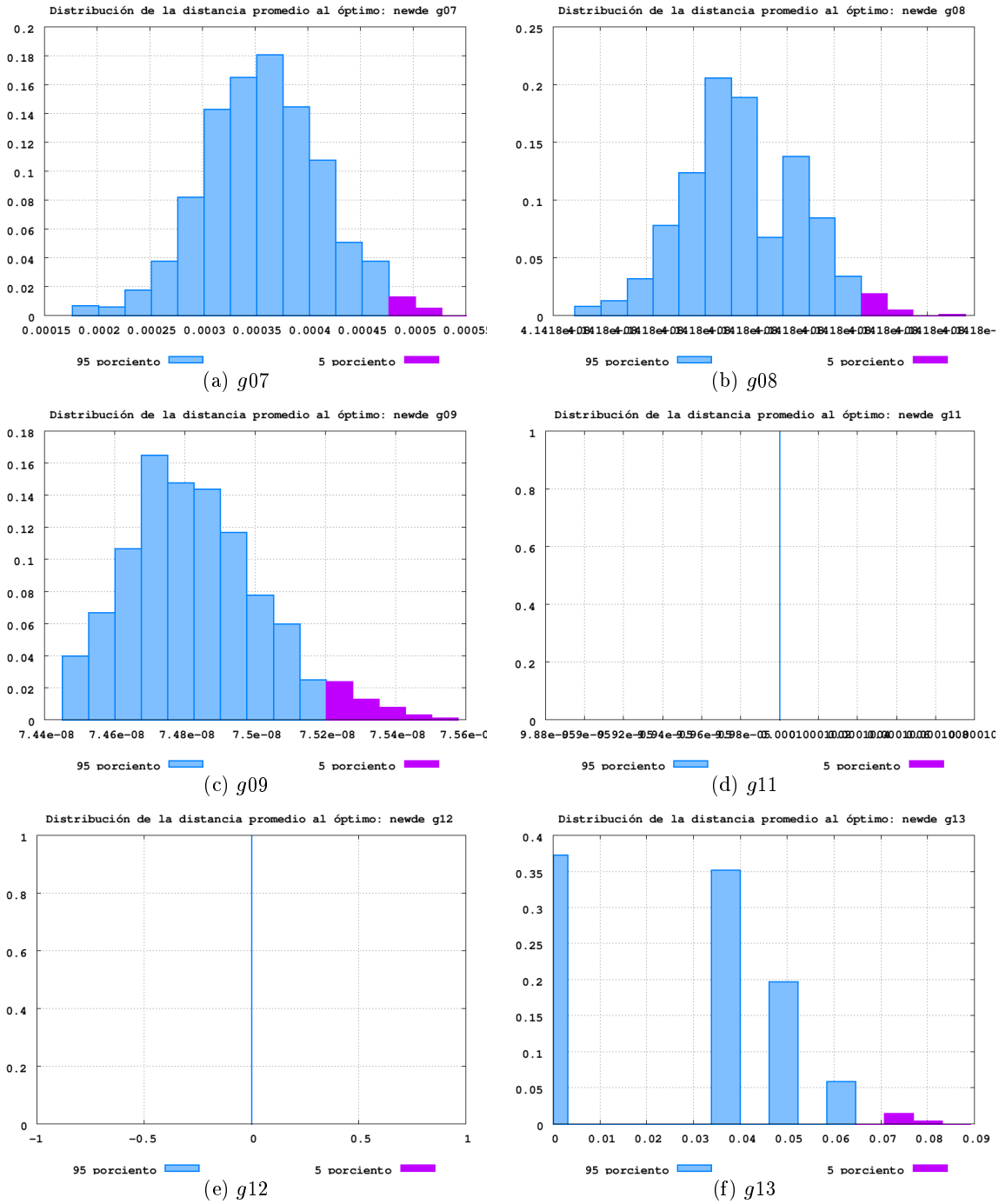


(a) $g01$ (b) $g02$

(c) $g03$ (d) $g04$

(e) $g05$ (f) $g06$

Figura C.4: Distribuciones de la distancia promedio al valor óptimo para el algoritmo **newde**



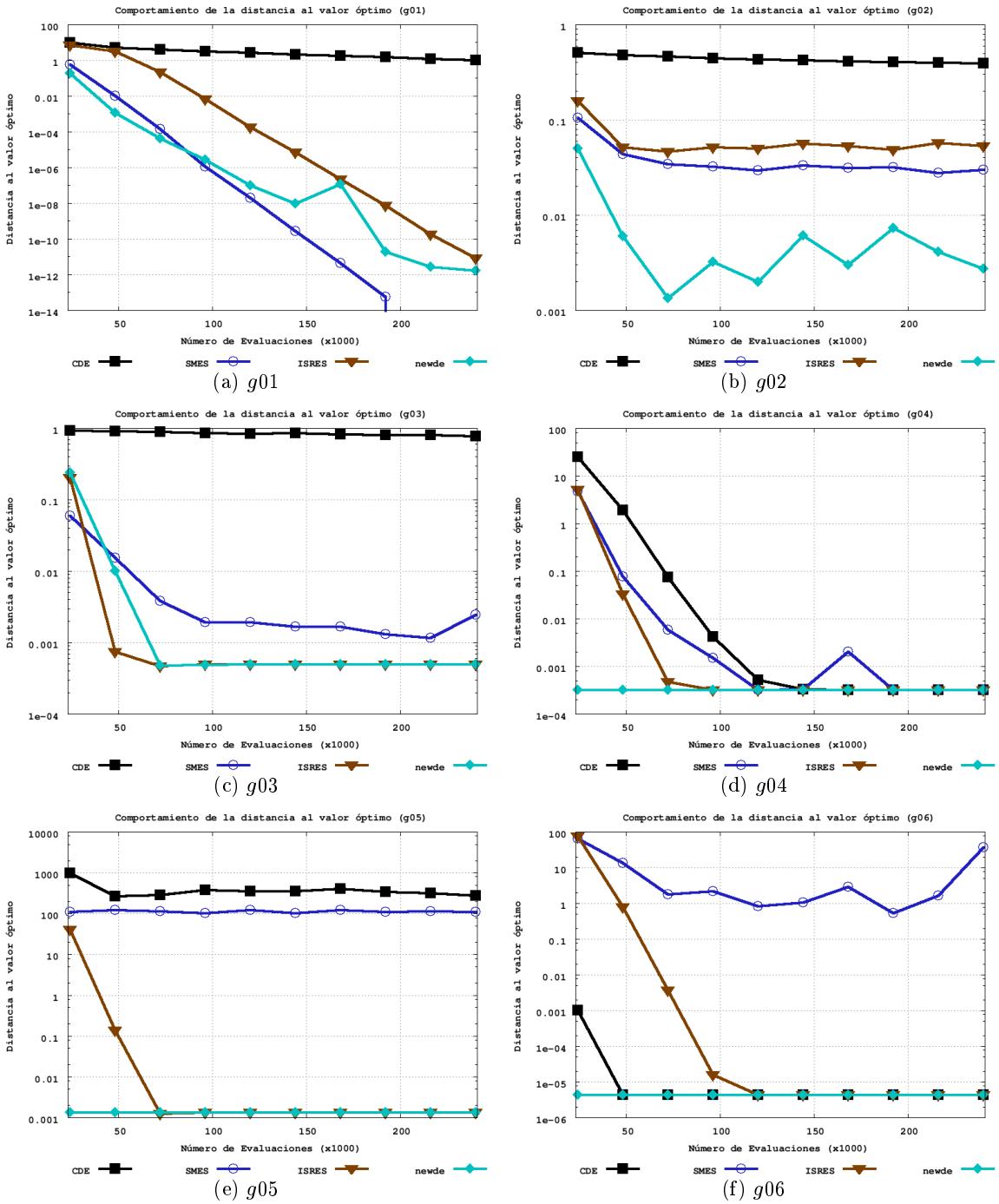
(a) $g07$ (b) $g08$

(c) $g09$ (d) $g11$

(e) $g12$ (f) $g13$

Figura C.5: Distribuciones de la distancia promedio al valor óptimo para el algoritmo **newde**

C.3. Gráficas de la Distancia al Valor Óptimo



(a) $g01$ (b) $g02$ (c) $g03$ (d) $g04$ (e) $g05$ (f) $g06$

Figura C.6: Gráficas de la distancia promedio al valor óptimo conforme se incrementa el número de evaluaciones de la función objetivo para los problemas $g01$, $g02$, $g03$,$g04$, $g05$, $g06$

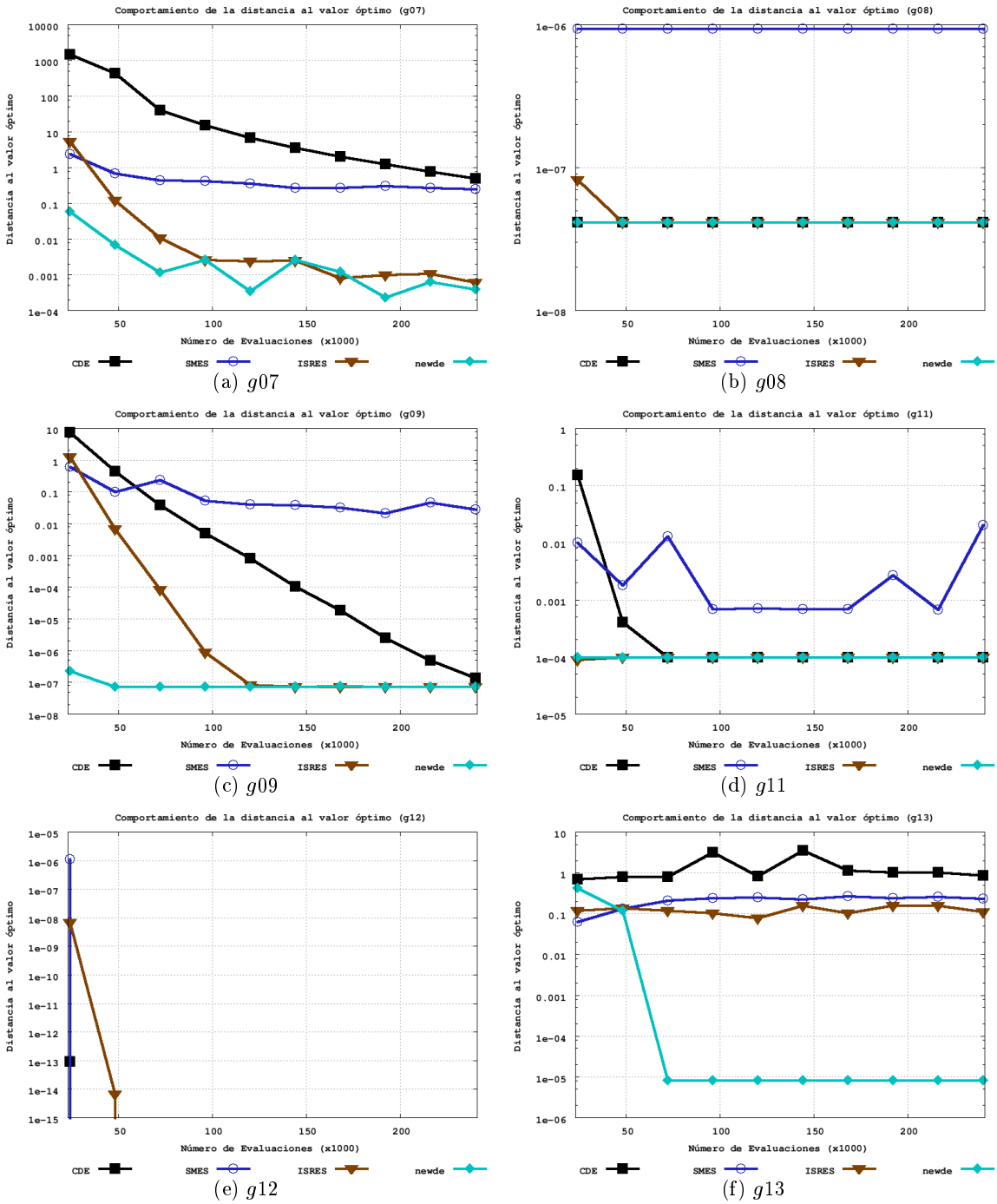


Figura C.7: Gráficas de la distancia promedio al valor óptimo conforme se incrementa el número de evaluaciones de la función objetivo para los problemas $g07$, $g08$, $g09$,$g11$, $g12$, $g13$

Bibliografía

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

[2] Ahlers, F. J., Carlo, W. D., and Fleiner, C. Differential Evolution (DE) for Continuous Function Optimization, 2005. Web page available at `http://www.icsi.berkeley.edu/~storn/code.html`.

[3] Beyer, H.-G., and Deb, K. On self-adaptive features in real-parameter evolutionary algorithms. *IEEE Transactions Evolutionary Computation 5*, 3 (2001), 250–270.

[4] Box. G. E. P. Evolutionary Operation: A method for increasing industrial productivity. *Applied Statistics*, 6 (1957), 81–101.

[5] Bäck, T., and Schwefel, H.-P. An Overview of Evolutionary Algorithms for Parameter Optimization. *Evolutionary Computation 1*, 1 (1993), 1–23.

[6] Coello, C. A. C. Theoretical and Numerical Constraint Handling Techniques used with Evolutionary Algorithms: A Survey of the State of the Art. *Computer Methods in Applied Mechanics and Engineering 191*, 11-12 (January 2002), 1245–1287.

[7] Coello, C. A. C., and Becerra, R. L. Constrained Optimization Using an Evolutionary Programming-Based Cultural Algorithm. In *Proceedings of the Fifth International Conference on Adaptive Computing in Design and Manufacture (ACDM'2002)* (University of Exeter, Devon, UK, April 2002), I. Parmee, Ed., vol. 5, Springer-Verlag, pp. 317–328.

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

[9] Coello Coello, C. A. The use of a multiobjective optimization technique to handle constraints. In *Proceedings of the Second International Symposium on Artificial Intelligence (Adaptive Systems)* (La Habana, Cuba, July 1999), A. A. O. Rodríguez, M. R. S. Ortiz, and R. S. Hermida, Eds., Institute of Cybernetics, Mathematics and Physics, Ministry of Science, Technology and Environment, pp. 251–256.

[10] Coello Coello, C. A. Constraint-handling using an evolutionary multiobjective optimization technique. *Civil Engineering and Environmental Systems 17* (2000), 319–346.

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

[12] Coello Coello, C. A., and Cruz Cortés, N. Use of Emulations of the Immune System to Handle Constraints in Evolutionary Algorithms. In *Intelligent Engineering Systems through Artificial Neural Networks (ANNIE'2001)* (St. Louis Missouri, USA, November 2001), C. H. Dagli, A. L. Buczak, J. Ghosh, M. J. Embrechts, O. Erson, and S. Kercel, Eds., vol. 11, ASME Press, pp. 141–146.

[13] Cohen, P. R. *Empirical methods for artificial intelligence.* MIT Press, Cambridge, MA, USA, 1995, ch. Computer-Intensive Statistical Methods, pp. 147–183.

[14] Coit, D. W., and Smith, A. E. Penalty guided genetic search for reliability design optimization. *Computers and Industrial Engineering 30*, 4 (September 1996), 895–904. Special Issue on Genetic Algorithms.

[15] Coit, D. W., Smith, A. E., and Tate, D. M. Adaptive Penalty Methods for Genetic Optimization of Constrained Combinatorial Problems. *INFORMS Journal on Computing 8*, 2 (June 1996), 173–182.

[16] Cruz-Cortés, N., Trejo-Pérez, D., and Coello, C. A. C. Handling Constraints in Global Optimization using an Artificial Immune System. In *Artificial Immune Systems. 4th International Conference, ICARIS 2005* (Banff, Canada, August 2005), C. Jacob, M. L. Pilat, P. J. Bentley, and J. Timmis, Eds., Springer, pp. 234–247. Lecture Notes in Computer Science Vol. 3627.

[17] Dasgupta, D. *Artificial Immune Systems and Their Applications.* Springer-Verlag, Berlin Heidelberg, 1999.

[18] Davis, L., Ed. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, New York, New York, 1991.

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

[20] Deb, K., and Agrawal, R. B. Simulated binary crossover for continuous search space. *Complex Systems 9* (1995), 115–148.

[21] Deb, K., Anand, A., and Joshi, D. A computationally efficient evolutionary algorithm for real-parameter optimization. *Evolutionary Computation 10*, 4 (2002), 371–395.

[22] Deb, K., Joshi, D., and Anand, A. Real-coded evolutionary algorithms with parent-centric recombination. In *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002* (2002), IEEE Press, pp. 61–66.

[23] Dorigo, M., and Caro, G. D. The Ant Colony Optimization Meta-Heuristic. In *New Ideas in Optimization*, D. Corne, M. Dorigo, and F. Glover, Eds. McGraw - Hill, 1999, pp. 11 – 61.

[24] Eiben, A. E., Aarts, E. H. L., and van Hee, K. M. Global Convergence of Genetic Algorithms: A Markov Chain Analysis. In *PPSN I: Proceedings of the 1st Workshop on Parallel Problem Solving from Nature* (London, UK, 1991), Springer-Verlag, pp. 4–12.

[25] Eiben, A. E., and Smith, J. E. *Introduction to Evolutionary Computing*. Springer-Verlag, 2003, ch. Evolution Strategies, pp. 71–87.

[26] Eshelman, L. J., and Schaffer, J. D. Real-coded genetic algorithms and interval-schemata. In *FOGA* (1992), pp. 187–202.

[27] Feoktistov, V., and Janaqi, S. Generalization of the Strategies in Differential Evolution. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS 2004), 2004, Santa Fe, New Mexico, USA* (New Mexico, USA, April 2004), IEEE Computer Society, p. 165a.

[28] Fogel, D. B. An Introduction to Simulated Evolutionary Optimization. *IEEE Transactions on Neural Networks 5*, 1 (January 1994).

[29] Fogel, D. B. *Evolutionary Computation: toward a new philosophy of machine intelligence*. IEEE Press, Piscataway, NJ, USA, 1995.

[30] Fogel, L., Owens, A., and Walsh, M. *Artificial Intelligence through Simulated Evolution*. Wiley, Chichester, UK, 1966.

[31] Glover, F. Scatter Search adn Path Relinking. In *New Ideas in Optimization*, D. Corne, M. Dorigo, and F. Glover, Eds. McGraw - Hill, 1999, pp. 297 – 316.

[32] Goldberg, D. E. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[33] Goldberg, D. E., and Vössner, S. Optimizing global-local search hybrids. In *Proceedings of the 1999 Genetic and Evolutionary Computation Conference (GECCO)* (1999), pp. 220–228.

[34] Hadj-Alouane, A. B., and Bean, J. C. A Genetic Algorithm for the Multiple-Choice Integer Program. *Operations Research 45* (1997), 92–101.

[35] Hajela, P., and Lee, J. Constrained Genetic Search via Schema Adaptation. An Immune Network Solution. In *Proceedings of the First World Congress of Stuctural and Multidisciplinary Optimization* (Goslar, Germany, 1995), N. Olhoff and G. I.Ñ. Rozvany, Eds., Pergamon, pp. 915–920.

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

[37] Hamida, S. B., and Schoenauer, M. ASCHEA: New Results Using Adaptive Segregational Constraint Handling. In *Proceedings of the Congress on Evolutionary Computation 2002 (CEC'2002)* (Piscataway, New Jersey, May 2002), vol. 1, IEEE Service Center, pp. 884–889.

[38] Hernández-Aguirre, A., Botello-Rionda, S., Coello Coello, C. A., Lizárraga-Lizárraga, G., and Mezura-Montes, E. Handling Constraints using Multiobjective Optimization Concepts. *International Journal for Numerical Methods in Engineering 59*, 15 (April 2004), 1989–2017.

[39] Herrera, F., and Lozano, M. A Taxonomy for the Crossover Operator for Real-Coded Genetic Algorithms: An Experimental Study. *International Journal of Intelligent Systems 18*, 3 (2003), 309–338.

[40] Herrera, F., Lozano, M., and Sánchez, A. M. Hybrid crossover operators for real-coded genetic algorithms: an experimental study. *Soft Computing 9*, 4 (2005), 280–298.

[41] Holland, J. H. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1992.

[42] Homaifar, A., Lai, S. H. Y., and Qi, X. Constrained Optimization via Genetic Algorithms. *Simulation 62*, 4 (1994), 242–254.

[43] Hooke, R., and Jeeves, T. A. "Direct Search" Solution of Numerical and Statistical Problems. *Journal of the ACM 8*, 2 (1961), 212–229.

[44] Jiménez, F., and Verdegay, J. L. Evolutionary techniques for constrained optimization problems. In *7th European Congress on Intelligent Techniques and Soft Computing (EUFIT'99)* (Aachen, Germany, 1999), H.-J. Zimmermann, Ed., Verlag Mainz. ISBN 3-89653-808-X.

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

[46] Joines, J., and Houck, C. On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with GAs. In *Proceedings of the first IEEE Conference on Evolutionary Computation* (Orlando, Florida, 1994), D. Fogel, Ed., IEEE Press, pp. 579–584.

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

[48] Kennedy, J., and Eberhart, R. C. The Particle Swarm: Social Adaptation in Information-Processing Systems. In *New Ideas in Optimization*, D. Corne, M. Dorigo, and F. Glover, Eds. McGraw - Hill, 1999, pp. 379 – 402.

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

[50] Kita, H. A comparison study of self-adaptation in evolution strategies and real-coded genetic algorithms. *Evolutionary Computation 9*, 2 (2001), 223–241.

[51] Kita, H., Ono, I., and Kobayashi, S. Multi-parental extension of the unimodal normal distribution crossover for real-coded genetic algorithms. In *Proceedings of the 1999 Congress on Evolutionary Computation (CEC99)* (1999), pp. 1581–1588.

[52] Koziel, S., and Michalewicz, Z. Evolutionary Algorithms, Homomorphous Mappings, and Constrained Parameter Optimization. *Evolutionary Computation 7*, 1 (1999), 19–44.

[53] Kuri-Morales, A., and Quezada, C. V. A Universal Eclectic Genetic Algorithm for Constrained Optimization. In *Proceedings 6th European Congress on Intelligent*

Techniques & Soft Computing, EUFIT'98 (Aachen, Germany, September 1998), Verlag Mainz, pp. 518–522.

[54] Lampinen, J. A Bibliography of Differential Evolution Algorithm, 2002. Web page available as `http://www2.lut.fi/~jlampine/debiblio.htm`.

[55] Lampinen, J. A Constraint Handling Approach for the Diferential Evolution Algorithm. In *Proceedings of the Congress on Evolutionary Computation 2002 (CEC'2002)* (Piscataway, New Jersey, May 2002), vol. 2, IEEE Service Center, pp. 1468–1473.

[56] Landa-Becerra, R., and Coello, C. A. C. Optimization with Constraints using a Cultured Differential Evolution Approach. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2005)* (New York, June 2005), H.-G. Beyer, U.-M. O'Reilly, and D. Arnold, Eds., vol. 1, Washington DC, USA, ACM Press, pp. 27–34. ISBN 1-59593-010-8.

[57] Le, T. V. A Fuzzy Evolutionary Approach to Constrained Optimization Problems. In *Proceedings of the Second IEEE Conference on Evolutionary Computation* (Perth, November 1995), IEEE, pp. 274–278.

[58] Lozano, M., Herrera, F., Krasnogor, N., and Molina, D. Real-coded memetic algorithms with crossover hill-climbing. *Evolutionary Computation 12*, 3 (2004), 273–302.

[59] Mezura-Montes, E. Uso de la Técnica Multiobjetivo NPGA para el Manejo de Restricciones en Algoritmos Genéticos. Master's thesis, Universidad Veracruzana, Xalapa, México, 2001. (In Spanish).

[60] Mezura-Montes, E. *Alternative Techniques to Handle Constraints in Evolutionary Optimization*. PhD thesis, Computer Science Section, Electrical Eng. Department., CINVESTAV-IPN, México City, México, December 2004.

[61] Mezura-Montes, E., and Coello, C. A. C. A Numerical Comparison of some Multiobjective-Based Techniques to Handle Constraints in Genetic Algorithms. Tech. Rep. EVOCINV-03-2002, Evolutionary Computation Group at CINVESTAV, Sección de Computación, Departamento de Ingeniería Eléctrica, CINVESTAV-IPN, México D.F., México, 2002. Available in the List of References on Constraint-Handling Techniques used with Evolutionary Algorithms at http://www.cs.cinvestav.mx/~constraint.

[62] Mezura-Montes, E., and Coello, C. A. C. Multiobjective-Based Concepts to Handle Constraints in Evolutionary Algorithms. In *Proceedings of the Fourth Mexican International Conference on Computer Science (ENC'2003)* (Los Alamitos, CA, September 2003), E. Chávez, J. Favela, M. Mejía, and A. Oliart, Eds., Apizaco, Tlaxcala, México, IEEE Computer Society, pp. 192–199.

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

[64] Michalewicz, Z., and Schoenauer, M. Evolutionary Algorithms for Constrained Parameter Optimization Problems. *Evolutionary Computation 4*, 1 (1996), 1–32.

[65] Montes, E. M., Coello, C. A. C., and Tun-Morales, E. I. Simple Feasibility Rules and Differential Evolution for Constrained Optimization. In *Proceedings of the 3rd Mexican International Conference on Artificial Intelligence (MICAI'2004)* (Heidelberg, Germany, April 2004), R. Monroy, G. Arroyo-Figueroa, L. E. Sucar, and H. Sossa, Eds., Springer-Verlag, pp. 707–716. Lecture Notes in Artificial Intelligence No. 2972.

[66] Moore, M., and McCabe, G. *Introduction to the Practice of Statistics.* Freeman, New York, 2004.

[67] Moscato, P. On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms. Tech. Rep. Caltech Concurrent Computation Program, Report. 826, California Institute of Technology, Pasadena, California, USA, 1989.

[68] Moscato, P. Memetic Algorithms: a Short Introduction. In *New Ideas in Optimization*, D. Corne, M. Dorigo, and F. Glover, Eds. McGraw - Hill, 1999, pp. 219 – 234.

[69] Nelder, J. A., and Mead, R. A simplex method for function minimization. *The Computer Journal 7* (1965), 308–313.

[70] Norman, B. A., and Bean, J. C. A Random Keys Genetic Algorithm for Job Shop Scheduling. Tech. Rep. 96-10, University of Michigan, Ann Harbor, 1996.

[71] Ono, I., and Kobayashi, S. A real-coded genetic algorithm for function optimization using unimodal normal distribution crossover. In *Proceedings of the 7th International Conference on Genetic Algorithms (ICGA)* (1997), pp. 246–253.

[72] O'Reilly, U.-M., and Oppacher, F. Hybridized crossover-based search techniques for program discovery. In *Proceedings of the 1995 World Conference on Evolutionary Computation* (Perth, Australia, January 1995), vol. 2, IEEE Press, pp. 573–578.

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

[74] Powell, M. J. D. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The Computer Journal 7*, 2 (1964), 155–162.

[75] Price, K. An Introduction to Differential Evolution. In *New Ideas in Optimization*, D. Corne, M. Dorigo, and F. Glover, Eds. McGraw - Hill, 1999, pp. 79 – 106.

[76] Rao, S. S. *Engineering Optimization: Theory and Practice*. John Wiley and Sons, Inc., New York, USA, 1996.

[77] Reynolds, R. G. Cultural Algorithms: Theory and Applications. In *New Ideas in Optimization*, D. Corne, M. Dorigo, and F. Glover, Eds. McGraw - Hill, 1999, pp. 367 – 377.

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

[79] Richardson, J. T., Palmer, M. R., Liepins, G., and Hilliard, M. Some Guidelines for Genetic Algorithms with Penalty Functions. In *Proceedings of the Third International Conference on Genetic Algorithms (ICGA-89)* (San Mateo, California, June 1989), J. D. Schaffer, Ed., George Mason University, Morgan Kaufmann Publishers, pp. 191–197.

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

[81] Runarsson, T. P., and Yao, X. Search Biases in Constrained Evolutionary Optimization. *IEEE Transactions on Systems, Man and Cybernetics 35*, 2 (May 2005), 233–243.

[82] Schoofs, L., and Naudts, B. Ant Colonies are Good at Solving Constraint Satisfaction Problems. In *Proceedings of the Congress on Evolutionary Computation 2000 (CEC'2000)* (Piscataway, New Jersey, July 2000), vol. 2, IEEE Service Center, pp. 1190–1195.

[83] Schwefel, H.-P. *Numerical Optimization of Computer Models*. John Wiley & Sons, Inc., New York, NY, USA, 1981.

[84] Schwefel, H.-P. P. *Evolution and Optimum Seeking: The Sixth Generation*. John Wiley & Sons, Inc., New York, NY, USA, 1993.

[85] Storn, R., and Price, K. Differential Evolution - a Simple and Efficient Adaptive Scheme for Global Optimization over Continuos Spaces. Tech. Rep. TR-95-012, International Computer Science Institute, March 1995.

[86] Storn, R., and Price, K. Differential Evolution a Simple and Efficient Heuristic for Global optimization over continuous spaces. *Journal of Global Optimization 4*, 11 (March 1997).

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

[88] Surry, P. D., Radcliffe, N. J., and Boyd, I. D. A Multi-Objective Approach to Constrained Optimisation of Gas Supply Networks : The COMOGA Method. In *Evolutionary Computing. AISB Workshop. Selected Papers* (Sheffield, U.K., April 1995), T. C. Fogarty, Ed., Springer-Verlag, pp. 166–180. Lecture Notes in Computer Science No. 993.

[89] Syswerda, G. Uniform Crossover in Genetic Algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms* (Virginia, USA, June 1989), J. D. Schaffer, Ed., Morgan Kaufmann, pp. 2–9.

[90] Teo, J. Differential Evolution with Self-Adaptive Populations. In *Proceedings of the 9th International Conference on Knowledge-Based Intelligent Information and Engineering Systems* (September 2005), pp. 1284–1290.

[91] Tsutsui, S., Yamamura, M., and Higuchi, T. Multi-parent recombination with simplex crossover in real coded genetic algorithms. In *Proceedings of the 1999 Genetic and Evolutionary Computation Conference (GECCO)* (1999), pp. 657–664.

[92] Yao, X., Liu, Y., Liang, K.-H., and Lin, G. Fast evolutionary algorithms. In *Advances in evolutionary computing: theory and applications*, G. Rozenberg, T. Bäck, and A. E. Eiben, Eds. Springer-Verlag New York, Inc., New York, NY, USA, 2003, pp. 45–94.

Y así termina este ciclo de nueve años de estudio.....