Instituto Politécnico Nacional.

Escuela Superior de Física y Matemáticas.

El algoritmo NSGA-II y su hibridación con un método basado en gradientes

Tesis que presenta

Andrés Alberto Becker Sanabria

para obtener el Titulo de

Licenciado en Física y Matemáticas

Directora de la Tesis

Dra. Adriana Lara López

Cd. de México **2016**

Índice general

Introducción v

1 Algoritmos genéticos 1

1.1 Métodos heurísticos 1

1.2 Conceptos básicos 5

1.3 Elementos de un AG 9

1.3.1 Parámetros del algoritmo 9

1.3.2 Representación genética 11

1.3.3 Función de aptitud y población inicial 12

1.4 Operadores genéticos 12

1.4.1 Selección por torneo 13

1.4.2 Operadores de cruza 14

1.4.3 Operador de mutación 19

1.4.4 Selección de supervivientes 24

1.5 Resultados numéricos 26

1.5.1 Función de Beale 26

1.5.2 Función de Rastrigin 27

1.5.3 Función de Rosenbrock 30

2 Algoritmo genético multiobjetivo: NSGA-II 35

2.1 Optimización multiobjetivo 35

2.1.1 Relación de dominancia 36

2.1.2 Optimalidad de Pareto 41

2.2 Algoritmo evolutivo multiobjetivo 46

2.2.1 Clasificación veloz de elementos no dominados 47

2.2.2 Estimación de la densidad de población 52

2.2.3 Indicador *crowding* 52

2.2.4 Torneo bajo *crowding* 55

2.2.5 Operadores de Selección y Cruza 58

2.2.6 Operador de Selección de Supervivientes 58

2.3 El algoritmo NSGA-II **61**
2.4 Evaluación del desempeño **62**
2.4.1 Indicadores 62
2.4.2 Ejemplo numérico 67

3 Algoritmo híbrido 71
3.1 Adaptación del buscador local **71**
3.1.1 Método de descenso con pendiente máxima 71
3.1.2 Adaptación del MDPM para dos funciones objetivo 75
3.2 Hibridación **79**
3.2.1 Resultados numéricos 80
3.2.2 Estimación de la eficiencia 87

4 Conclusiones 91

A Funciones de prueba biobjetivo 95

Lista de Figuras 98

Lista de Tablas 100

Lista de Algoritmos 101

Lista de Acrónimos 103

Introducción

Para las ciencias de la computación y la ingeniería, la naturaleza ha sido siempre una fuente de inspiración. En una primera instancia se intentó imitar las formas y estructuras naturales para el diseño de artefactos; así como de emular, en cierta medida, las capacidades del cerebro humano (esto último dio origen en la segunda mitad del siglo pasado a la llamada *Inteligencia Artificial*). Un acercamiento más ambicioso consistió en imitar los procesos evolutivos; partiendo de la base de que éstos moldearon tanto el mismo cerebro humano como las formas y organización de los seres vivos, conformando complejos (eco)sistemas. En sus orígenes, el análisis formal de algoritmos de cómputo dio a los científicos cierta certeza de prever las capacidades y limitaciones de sus métodos, así como de estimar en cuánto tiempo llegarían a resolver un problema si este era el caso. Con el paso del tiempo, y motivados por el acelerado crecimiento de la eficiencia en la maquinaria de cómputo, se han propuesto modelos más complejos para tratar los problemas de la vida real; dichos modelos no necesariamente pueden ser analizados, de manera formal, más que en cierto número limitado de casos. De aquí surgen los métodos heurísticos, y en particular los llamados *algoritmos genéticos.*

Los algoritmos genéticos son métodos de aproximación de soluciones. Mediante la imitación de algunos de los procesos naturales que han dado origen a la evolución de las especies, dichos algoritmos buscan emular, en cierta medida, la eficacia del proceso evolutivo para obtener soluciones aceptables, a problemas complejos, en un tiempo razonable. Hoy en día es común encontrar aplicaciones de este tipo de algoritmos en casi todos los ámbitos de la tecnología y las ciencias. Aún se considera un área abierta de investigación la mejora de dichos algoritmos mediante la adaptación de métodos matemáticos que ayuden a la eficiencia de sus operadores.

Los objetivos de este trabajo son *(i)* presentar de manera general los mecanismos que conforman los algoritmos genéticos para aproximar soluciones a problemas de optimización, con una y dos funciones objetivo y *(ii)* proponer la adaptación de un método clásico de Programación Matemática, para problemas con dos funciones objetivo, dentro el algoritmo genético. En la tesis se presentan los resultados numéricos de diversos experimentos para ejemplificar y analizar de manera experimental el método propuesto.

En el capítulo uno se estudia el algoritmo genético clásico. Después de una breve introducción a las técnicas heurísticas se establecen los conceptos y la nomenclatura necesaria para trabajar con este tipo de algoritmos. También se estudia el com-

portamiento del algoritmo mediante experimentos numéricos, sobre problemas de optimización con una sola función objetivo. En el capítulo dos, se presenta el problema de optimización multiobjetivo, así como el concepto de dominancia y de óptimo de Pareto. Se estudian los detalles para la adaptación del algoritmo genético a este tipo de problemas y se presentan ejemplos numéricos. En el capítulo tres, se propone una manera de hibridizar el algoritmo genético con una adaptación del método de Cauchy (descenso con pendiente máxima) para problemas con dos funciones objetivo. Posteriormente se muestra un estudio experimental sobre una serie de problemas de prueba para observar el comportamiento de dicha propuesta. Finalmente se incluyen las conclusiones y el trabajo a futuro, resultado de esta investigación; así como dos apéndices.

Capítulo 1

Algoritmos genéticos

1.1. Métodos heurísticos

Etimológicamente la palabra *heurística* proviene del griego "heurísk" y significa encontrar o inventar.[1] Otra definición podría ser

> *"Forma de buscar la solución a un problema mediante métodos no rigurosos, por tanteo, reglas empíricas, etc.".*

En las ciencias de la computación generalmente se busca 1) construir algoritmos con buenos tiempos de ejecución y 2) que resuelvan el problema arrojando buenas soluciones, usualmente las óptimas. Una heurística es un algoritmo que se aleja de uno o ambos de estos objetivos, pues podría encontrar una muy buena solución, en un tiempo de ejecución corto, pero sin tener pruebas de que es la óptima o sin garantizar que no existe una forma más eficiente de llegar a dicha solución.

Las *Metaheurísticas* son métodos heurísticos de alto nivel, es decir, son algoritmos diseñados para resolver una clase de problemas numéricos en general. De ahí el prefijo *meta* que significa "más allá", o en este caso "nivel superior". Las metaheurísticas generalmente usan parámetros dados por el usuario, dependientes del problema, y se consideran métodos "genéricos". Por ejemplo, el *Algoritmo Genético* (AG), propuesto por John H. Holland en 1975, que dio lugar a uno de los paradigmas principales de los *Algoritmos Evolutivos* (AEs).

El origen de los AEs se remonta a 1950, cuando Alan Turing propuso una "Máquina de Aprendizaje" que simularía los principios de la evolución [Tur50]. Luego en 1954 comenzó la simulación computacional de la evolución con el trabajo de Nils Aall Barricelli [Bar62], quien usaba la computadora del Instituto para Estudios Avanzados de la Universidad de Princeton; sin embargo, sus publicaciones pasaron mayormente inadvertidas. A comienzos de 1957, el genetista Australiano Alex Fraser publicó una serie de artículos en simulación de "selección artificial" de organismos [Fra60]. Desde entonces, la simulación por computadora de la evolución se hizo cada vez

[1] Esta etimología se comparte con "Eureka" mencionado por Arquímedes al encontrar la solución al problema de comprobar si la corona del rey era de oro.

más común. En la década de 1960, Hans-Joachim Bremermann publicó una serie de artículos que incluían soluciones a problemas de optimización, los cuales consideraban mecanismos de recombinación, mutación y selección. El trabajo de Bremermann estableció las bases de los AGs modernos. Para 1963, Barricelli reporto que había logrado simular la habilidad de jugar un juego simple. De este modo, la *evolución artificial* fue reconocida mundialmente como un método para aproximar soluciones a problemas de optimización, pues entre 1960 y 1970 Ingo Rechenberg y Hans-Paul Schwefel fueron capaces de resolver un complejo problema de ingeniería mediante un método que denominaron Estrategia Evolutiva (EE).

Sin embargo, AGs se volvieron populares gracias al trabajo de John Holland, pues a principios de la década de 1970 él y sus alumnos de la Universidad de Michigan investigaron los *Autómatas Celulares*, lo cual introdujo y formalizó el marco de trabajo para los AG. Todo esto fue incluido en 1975 en su famoso libro [Hol75], de donde destaca el *Teorema de los esquemas* que buscaba fundamentar de manera teórica los principios detrás del buen funcionamiento de este tipo de algoritmos. La investigación en AGs permaneció principalmente en ideas y teorías hasta mediados de la década de 1980, cuando se llevó a cabo la primera Conferencia Internacional en Algoritmos Genéticos[2].

Los AGs han sido utilizados para resolver problemas de aplicación en diversas disciplinas, de entre los que podemos destacar: diseño de antenas para satélites [CRLL12], diseño automatizado de sistemas mecatrónicos (NSF) [SFH+03], aprendizaje de un robot en su comportamiento usando AG [GH88], problemas de transporte [BA03], diseño automático e investigación de materiales compuestos [MSRM93], diseño multi-objetivo para optimizar auto partes en pruebas de choque, ahorro de peso, entre otros [LLY+08], biología y química computacional [GJ06], diseño automatizado de sistemas de negociación sofisticados en el sector financiero [Che12], granjas de servidores y "Data Center" [AMYA12], diseño de circuitos electrónicos, conocido como Hardware evolvable [HIK+96], por mencionar algunos.

Para clarificar el por qué del uso de los métodos heurísticos en la resolución de problemas de optimización, damos a continuación uno de los ejemplos más simples y representativos de un problema que no es posible resolver por métodos convencionales, con la capacidad computacional actual.

Ejemplo 1.1. *(Problema del Agente Viajero) Considere un vendedor que debe visitar todas las ciudades en su ruta de ventas, una y solo una vez, y luego debe regresar a su casa recorriendo la menor distancia posible. Si definimos la distancia entre la ciudad i y la ciudad j como $dist(i,j)$, y además suponemos que $dist(i,j)$ es simétrica*[3]*,*

[2]Actualmente la conferencia internacional más grande que se celebra en estos temas es la denominada GECCO, la cual es auspiciada anualmente por la *Association for Computing Machinery* (misma asociación que otorga el *Turing Award*, que es el máximo premio otorgado en las Ciencias de la Computación).

[3]Es decir que $dist(i,j) = dist(j,i)$ para cualquier par de ciudades i, j que estén conectadas entre sí.

entonces ¿cómo debe el vendedor planear su ruta para que la distancia total recorrida sea mínima?

Algoritmo 1: Ascenso por pasos [ZM04].

Input: Elementos de la región factible $v \in \mathcal{F}$
Output: Un óptimo local $v_o \in \mathcal{F}$

```
1  t = 0
2  Iniciamos v_o
3  while t ≤ MAX do
4  |  local=FALSO
5  |  Seleccionamos un punto al azar v_c ∈ F
6  |  Calculamos val(v_c)
7  |  while local=VERDADERO do
8  |  |  Evaluamos cada punto de la vecindad de v_c y seleccionamos el punto
   |  |   con el mejor valor v_n
9  |  |  if val(v_n) es mejor que val(v_c) then
10 |  |  |  val(v_c) = val(v_n)
11 |  |  end
12 |  |  else
13 |  |  |  local=VERDADERO
14 |  |  end
15 |  end
16 |  t = t + 1
17 |  if val(v_c) es mejor que val(v_o) then
18 |  |  v_o = v_c
19 |  end
20 end
21 return v_o
```

La figura 1.1 muestra un problema de este tipo para 20 ciudades, note que la distancia entre la ciudad i y la ciudad j, para $i, j \in \{1, \ldots, 20\}$, no se muestran en la figura, sin embargo, puede suponer que es conocida y que además es simétrica.

Entonces, ¿cuántas posibles soluciones tenemos a nuestro problema? Una respuesta a esta pregunta podría ser ver el espacio de soluciones como todas las posibles combinaciones o rutas entre las 20 ciudades. Es decir, para la primera ciudad tenemos 20 posibilidades, para la segunda 19, la tercera 18 y así sucesivamente, por lo que el número total de posibles combinaciones es 20!. Cada combinación mostraría una secuencia de las ciudades a ser visitadas, como 2-15-19-1-6-9-4-...-18, comenzando y terminado en la casa del vendedor. Sin embargo, note que rutas como:

2 - ... - 6 - 15 - 3 - 11 - 19 - 17,
15 - 3 - 11 - 19 - 17 - 2 - ... - 6,
3 - 11 - 19 - 17 - 2 - ... - 6 - 15,

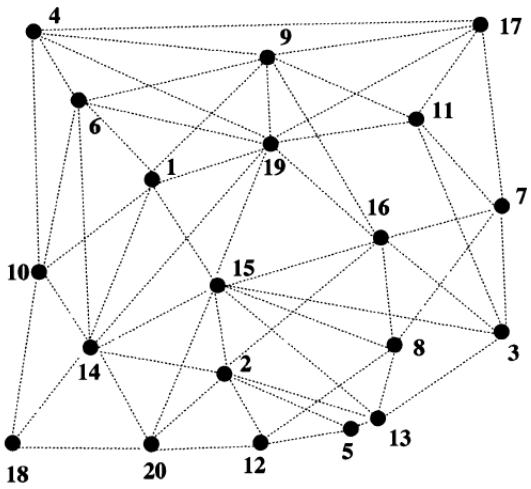


Figura 1.1: *Ejemplo para 20 ciudades, con costo de traslado entre cada par simétrico y conocido [ZM04].*

etc.

en realidad son las mismas, ya que éstas definen el mismo circuito, por lo que cada ruta puede ser expresada de 20 distintas formas. Esto reduce nuestro espacio de soluciones de 20! a $20!/20 = 19!$. Más aún, como suponemos que nuestro problema es simétrico, entonces las rutas:

5 - 3 - 11 - 19 - 17 - 2 - ... - 6,
6 - ... - 2 - 17 - 19 - 11 - 3 - 5

son las mismas, pues representan el mismo costo. Debido a esto y a la observación anterior, cada ruta puede ser expresada de $2 * 20$ formas distintas. Por ello nuestro espacio de posibles elementos tiene un total de:

$$20!/(2 * 20) = 19!/2$$

En general para un problema de este tipo con n ciudades a visitar tenemos $(n-1)!/2$ posibles rutas.

Para representar la forma en la que $(n-1)!/2$ crece con respecto a n, considere los siguientes números:

- Para $n = 20$ ciudades tenemos alrededor de 60'822'550'200'000'000 posibles soluciones.
- Para $n = 50$ ciudades tenemos alrededor de 15'207'046'600'000'000'000'000' 000'000'000'000'000'000'000'000'000'000'000'000'000'000'000'000 posibles soluciones.

Si definimos el costo como la suma de cada distancia entre ciudades consecutivas en la ruta, por ejemplo

15 - 3 - 11 - 19 - 17 - 2 - ... - 6

entonces

$$cost = dist(15, 3) + dist(3, 11) + dist(11, 19) + ... + dist(6, 15).$$

Luego, para nuestro problema de 20 ciudades, si quisiéramos encontrar la ruta de costo mínimo y pudiéramos determinar computacionalmente el costo de 100 rutas por segundo, entonces tardaríamos alrededor de 1'928'670'415 años o casi 2 millones de milenos en encontrar la solución, y para ese entonces probablemente ya ni siquiera existamos. Entonces, ¿cómo podríamos encontrar al menos una ruta que se acerque a la óptima en un tiempo razonable? A continuación, se explica el método de ascenso de colinas (*hill climbing*), el cual puede ayudarnos a encontrar una ruta óptima, al menos en un subconjunto de nuestra región factible, es decir, un óptimo local.

Los métodos de ascenso de colinas usan una técnica de mejora iterativa, existen algunos algoritmos, los cuales difieren principalmente en la forma en la que una nueva solución es selecciona para ser comparada con la solución actual. Una de las versiones más simples de estos métodos es el de *"ascenso por pasos"*, el cual mostramos en el algoritmo 1. Para empezar, seleccionamos un punto al azar de la región factible v_c, luego todos los puntos en una vecindad dada de v_c son considerados y el punto v_n que regrese el mejor valor $val(v_n)$ es confrontado contra $val(v_c)$. Si $val(v_n)$ es mejor que $val(v_c)$, entonces, v_n se convierte en v_c. De lo contrario, ninguna mejora local es posible y el algoritmo ha encontrado un óptimo local. En ese caso, el algoritmo se repite con un nuevo punto de la región factible seleccionado al azar.

El algoritmo 1 es un ejemplo muy sencillo de un método heurístico aplicado a un problema de optimización que no es posible resolver para n demasiado grande, ya que nos otorga una solución alternativa en el tiempo que nosotros decidamos, sin embargo, nada nos garantiza que esta solución es el óptimo global y no un óptimo local.

1.2. Conceptos básicos

En Inteligencia Artificial (IA), un AG es una técnica heurística estocástica[4] que pertenece a la extensa clase de los AE, los cuales aproximan soluciones a problemas de optimización y no necesita información específica para guiar su búsqueda. Presentan analogías con la teoría biológica de la evolución, ya que se basan en la supervivencia del más apto además de integrar propiedades como la herencia, mutación y cruza [Coe99].

La idea general sobre la que se desarrollan los AGs es la siguiente. Dado un *entorno o ambiente* éste es llenado con una población de *individuos*, los cuales compiten

[4]Un AG es estocástico cuando el valor de ciertas propiedades de los individuos o posibles soluciones aproximadas por el algoritmo son determinados de forma aleatoria (como probabilidad de cruce, mutación, etc.).

Algoritmo 2: Pseudocódigo de un AG simple.

```
1  begin
2      Generar una población inicial aleatoria
3      Evaluar la aptitud de cada individuo
4      while Condición de finalización no se satisfaga do
5          Seleccionar padres de la población
6          Producir hijos con los padres seleccionados
7          Mutar a hijos
8          Extender la población añadiendo a los hijos
9          Seleccionar individuos para la siguiente población
10     end
11 end
```

entre sí por la *supervivencia* y la *reproducción*. El éxito de dicha competencia dependerá de la *aptitud* o *adaptabilidad* de los individuos, la cual, está determinada por el entorno. En términos de solución de problemas mediante "prueba y error", tenemos un grupo de posibles soluciones (los individuos de la población), las cuales pueden ser generadas de forma aleatoria para la primera generación. La calidad de dichas soluciones (aptitud) dependerá de que tan bien resuelven el problema. Si la calidad de una solución es buena en comparación con los demás elementos de la población, entonces esta tendrá mejores posibilidades de ser seleccionada para construir futuras soluciones (reproducirse) e incluso de mantenerse en la siguiente generación de individuos.

Definición 1.1. *Sea $f : \mathbb{R}^n \to \mathbb{R}$, un Problema de Optimización es aquel que tiene la siguiente forma:*

$$\begin{array}{lll} \textit{Minimizar} & f(x) & \\ \textit{Sujeto a} & g_j(x) \geq 0, & j = 1, 2, ..., J; \\ & h_q(x) = 0, & q = 1, 2, ..., Q; \\ & x_i^L \leq x_i \leq x_i^S, & i = 1, 2, ..., n; \end{array} \tag{1.1}$$

donde $x^L, x^S \in \mathbb{R}^n$.

A la función f la llamaremos función objetivo, mientras que a las $J+Q$ funciones de la forma $g(x) \geq 0$ y $h(x) = 0$ las llamaremos funciones restrictivas o simplemente restricciones. Las n desigualdades de la forma $x^L \leq x \leq x^S$ son llamadas restricciones de caja.

Definición 1.2. *Sea $\mathcal{D} \subseteq \mathbb{R}^n$. Definimos la Región Factible para un problema de optimización, como el conjunto de vectores $x \in \mathbb{R}^n$ tales que cumplen las restricciones de la definición 1.1, es decir:*

$$\mathcal{D} := \{x \in \mathbb{R}^n \mid g_j(x) \geq 0,\ h_q(x) = 0 \ y\ x^L \leq x \leq x^S \ \forall\ j = 1, \ldots, J,\ q = 1, \ldots, Q\}$$

A los vectores $x \in \mathcal{D}$ los llamaremos soluciones factibles.

Se le llama *generación* al conjunto de individuos, o posibles soluciones, sobre los cuales el AG operara en la iteración que se encuentre. En cada generación la aptitud de los individuos de la población es evaluada de manera individual. Usualmente, la aptitud es el valor de la *función objetivo* en el problema de optimización.

Para hacer *evolucionar* a los individuos de la población inicial, estos son sometidos a operadores de selección, cruza y mutación con el fin de producir un conjunto de hijos. Por último, la siguiente generación de individuos es formada escogiendo los individuos más aptos de entre los padres y los hijos. De este modo, una nueva población de individuos es seleccionada para ser usada en la siguiente iteración del algoritmo. Comúnmente, el algoritmo termina cuando un número máximo de generaciones es producido, o un nivel de adaptación satisfactoria ha sido alcanzada.

La relación "Evolución - Problema" se muestra de forma más sencilla en la tabla 1.1.

Evolución		Resolución de problema
Ambiente	$\Longleftrightarrow$	Problema
Individuos	$\Longleftrightarrow$	Posibles soluciones
Adaptabilidad	$\Longleftrightarrow$	Calidad de solución

Tabla 1.1: *Relación entre un AE básico y la evolución natural orientada a la solución de problemas [ES03].*

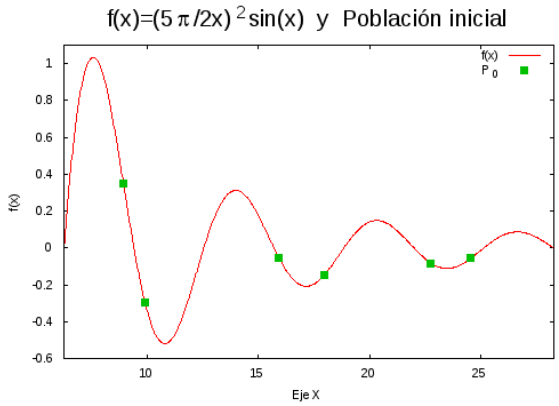
Para ilustrar un poco mejor lo que es un AG y cómo funciona mostramos un pequeño ejemplo a continuación.

Ejemplo 1.2.

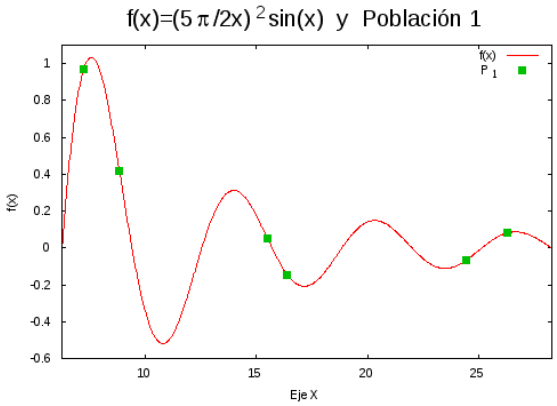
$$\begin{aligned} Maximizar \qquad & f(x) = \left(\frac{5\pi}{2x}\right)^2 sen(x); \\ Sujeto\ a \qquad & 2\pi \leq x \leq 9\pi. \end{aligned} \tag{1.2}$$

Claramente es posible encontrar el máximo de esta función por medio de métodos convencionales, sin embargo, el objetivo es meramente ilustrativo. Para este ejemplo, se generó de manera aleatoria una población de 6 individuos dentro del rango $2\pi \leq x \leq 9\pi$. Luego, se llevo a cabo el proceso de selección, cruza y mutación de los individuos en la población inicial para obtener la segunda generación. Este procedimiento fue realizado hasta que todos los individuos de la población estaban próximos al óptimo global (es decir, 7 iteraciones o 7 generaciones). Este proceso puede ser apreciado en el grupo de figuras 1.2, el cual muestra la evolución de la población mediante las iteraciones del AG.

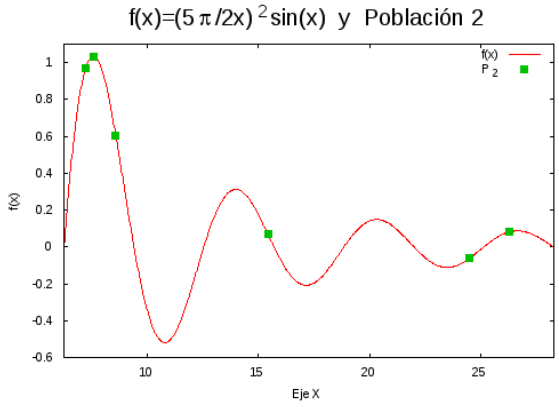
Para abordar este ejemplo, se adaptó el código de un AG desarrollado por el Prof. Kalyanmoy Deb del Laboratorio de Algoritmos Genéticos de Kanpur (KanGAL por



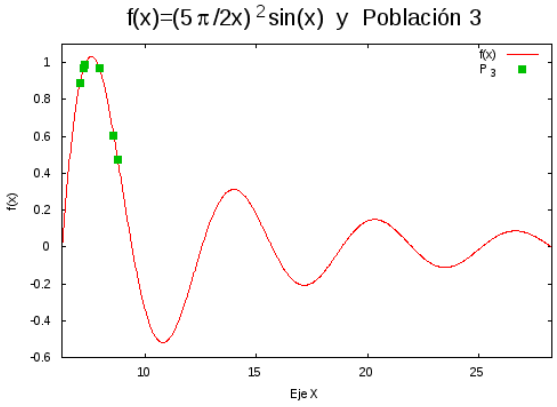
(a) $f(x)$ y Población inicial



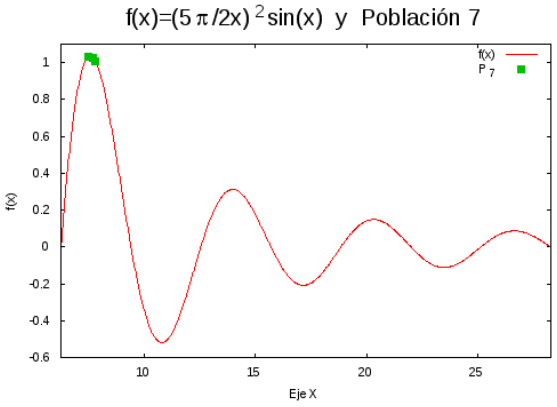
(b) $f(x)$ y Población 1.



(c) $f(x)$ y Población 2.



(d) $f(x)$ y Población 3.



(e) $f(x)$ y Población 7.

Figura 1.2: *Aproximación de la población inicial al óptimo (máximo) global del ejemplo 1.2 con 7 generaciones.*

sus siglas en inglés)[5]. El algoritmo 2 muestra de forma más explícita y clara el conjunto de procesos que conforman un AG básico.

1.3. Elementos de un AG

En general un AG típico requiere lo siguiente 5 elementos básicos para poder operar sobre un problema:

1. Valores para los *parámetros* del AG como: tamaño de la población, probabilidad de cruza y mutación, número máximo de generaciones, etc.
2. Una *representación genética* de los elementos del conjunto de posibles soluciones.
3. Una *población inicial* de soluciones potenciales o individuos (usualmente generada de forma aleatoria, aunque también puede ser formada con métodos determinísticos).
4. Una *función de aptitud* que juegue el papel de ambiente para evaluar los elementos del conjunto de posibles soluciones.
5. Operadores que realicen la *selección de padres, cruza, mutación* y *selección de supervivientes*[6].

Notación. *Definimos a la población P en la generación i como:*

$$P_i := \{x_i^j \mid x_i^j \in \mathbb{R}^n,\ j = 1, 2, ..., \lambda,\ i = 0, 1, 2, ..., G\}$$

donde x_i^j es el individuo j en la generación i.

1.3.1. Parámetros del algoritmo

En la tabla 1.2 definimos los parámetros necesarios para que un AG como el mostrado en algoritmo 2 funcione.

Los parámetros que un AG necesita para funcionar dependerá del diseño de sus operadores. Además de indicar el número máximo de generaciones, n también indica el numero iteraciones o veces que el algoritmo evolucionara la población inicial y es el *criterio de finalización* del AG (ver la línea 4 del pseudocódigo del algoritmo 2), es decir, el algoritmo se detiene después de que la población P_n fue producida.

El parámetro λ especifica el tamaño de la población o número de posibles soluciones sobre las que AG operara en cada iteración. Existen AG en los que el tamaño

[5]Kalyanmoy Deb es profesor del Instituto de Tecnología de Kanpur, India. Parte del código que se usó en este capítulo fue adaptado de un algoritmo codificado en C que se encuentra disponible en http://www.iitk.ac.in/kangal/codes.shtml

[6]El Operador de selección de supervivientes se encarga de seleccionar a los individuos más aptos de entre las poblaciones de padres y descendientes para formar la nueva generación de individuos.

Parámetro	Descripción
G	Numero de generaciones
λ	Tamaño de la población
p_c	Probabilidad de cruza
p_m	Probabilidad de mutación
γ	Semilla aleatoria
η	Indice de distribución de cruza
ν	Indice de distribución de mutación

Tabla 1.2: *Parámetros usados en un AG.*

de la población λ varia de generación en generación, sin embargo, el AG que usamos mantiene a λ siempre constante, además λ debe ser un numero par mayor o igual que 2.

La probabilidad de cruza y mutación, p_c y p_m respectivamente, son las responsables de decidir si los operadores de cruza y mutación son ejecutados una vez por individuo en cada generación. Es decir, dados 2 individuos elegidos por el operador de selección, x_i^1 y x_i^2 con $i \in \{1, ..., G\}$, y un número aleatorio $u \in (0, 1)$, si $u \leq p_c$ entonces el operador de cruza será ejecutado sobre x_i^1 y x_i^2. De lo contrario, x_i^1 y x_i^2 no son cruzados y pasan directamente al siguiente operador. Análogamente para el operador de mutación si $u \leq p_m$ entonces, x_i^j será mutado, para $i \in \{1, ..., G\}$ y $j \in \{1, ..., \lambda\}$.

La semilla aleatoria γ es el parámetro bajo el cual se genera los números aleatorios sobre los que operara el AG durante toda su ejecución. La función que utilicemos para generar dichos números aleatorios dependerá de γ y será de vital importancia para el AG, ya que esta garantizará que obtengamos siempre los mismos resultados al ejecutar el algoritmo con los mismos parámetros. La convergencia del AG y/o la rapidez con la cual aproxima una solución aceptable, depende hasta cierto punto de γ, ya que de este parámetro dependen los operadores de cruza, mutación y selección, o más importante aún, la población inicial (pues es generada aleatoriamente), ya que este último se encarga de permear la región factible del problema de optimización.

Los parámetros η y ν son usados por el operador de cruza y el operador de mutación respectivamente, y será explicado con detalle más adelante.

Los parámetros utilizados en el ejemplo 1.2 pueden ser consultados en la tabla 1.3.

Parámetro	G	λ	P_c	P_m	γ	η	ν	C_I	C_S
Valor	7	6	0.8	0.4	0.321	2	100	2π	2π

Tabla 1.3: *Parámetros usados en el ejemplo 1.2.*

donde C_I y C_S son la cota inferior y superior de x, respectivamente (ecuación 1.2), que si bien no se consideran parámetros, las especificamos ya que son utilizadas por el operador encargado de generar la población inicial y los operadores de cruza y

mutación. La correcta elección de éstos es de vital importancia para que el algoritmo aproxime buenas soluciones. Los criterios bajo los cuales se escogen, así como de su sensibilidad a los cambios, serán explicados y clarificados a medida que analicemos cada operador y desarrollamos diferentes ejemplos.

1.3.2. Representación genética

En biología, la genética es la ciencia que estudia la forma en la que las características de un organismo o individuo se representan y transmiten de generación en generación. Dichas características están codificadas en el *genotipo* del individuo. Por medio de la cruza, los individuos transmiten su información genética a los descendientes; dependiendo del ambiente u entorno, algunas características codificadas en el genotipo pueden o no manifestarse. A las características que se manifiestan se les denomina *fenotipo*.

Para que un AG funcione, es necesario crear una "conexión" entre el contexto del problema original y el espacio de posibles soluciones, ya que es en este último donde la evolución tomará lugar. En términos simples, es necesario codificar las características de cada individuo (su fenotipo), en un lenguaje sobre el cual podamos trabajar, es decir, el genotipo. La relación que existe entre el fenotipo y su respectivo genotipo es precisamente la *representación*. Existen distintos tipos de representación, entre las cuales destacan la *binaria* y la *real*. Holland formalizó el marco de trabajo para los AG (1975) y lo hizo a través de la representación binaria [Hol75]. Desde entonces ésta ha sido una de las más estudiadas y utilizadas.

Por ejemplo, dado un problema de optimización entera, $\mathbb{Z}$ conformaría el conjunto de posibles soluciones, luego si escogiéramos 77 como una solución entonces, 77 sería el fenotipo y 1001101 su respectivo genotipo.

Sin embargo, la representación que utilizaremos a lo largo de este trabajo será la real. Ya que, si bien la binaria realiza una buena analogía entre la codificación genética de los organismos en biología y las posibles soluciones en problemas de optimización, ésta presenta algunas desventajas en comparación con la representación real. Por ejemplo, la representación binaria no mapea adecuadamente el espacio de búsqueda en el de representación, ya que dados los enteros 7 y 8, sus respectivas representaciones son 0111 y 1000, las cuales, difieren en los 4 bits. Al fenómeno anterior se le conoce cono *Risco de Hamming*[Coe99]. Otro problema que se suscita al trabajar con representación binaria es cuando se intenta resolver problemas con demasiadas variables y una buena precisión para cada una de ellas, ya que las cadenas binarias se volverán demasiado largas y el desempeño del AG será pobre [Coe99].

La representación real es bastante más sencilla que la binaria. En el ejemplo 1.2, x_i^j representa al individuo j de la población i, luego si x_i^j fuese igual a 15.92052, este número representaría el genotipo, mientras que la función objetivo valuada en x_i^j, $f(x_i^j) = 0.06675$ seria el fenotipo, es decir, la aptitud.

1.3.3. Función de aptitud y población inicial

La función de aptitud es la encargada de medir la calidad de los individuos o posibles soluciones en el problema. Usualmente ésta es la misma que la función objetivo del problema de optimización, aunque en AG más especializados es usual encontrar funciones de aptitud más elaboradas. Sin embargo, éstas siempre dependerán de la función objetivo. Para esta sección, la función de aptitud que usamos coincide con la función objetivo de nuestro problema de optimización.

Como se mencionó anteriormente, la población inicial P_0 es generada aleatoriamente, a no ser que se tenga información previa del problema y se deseé usar una ya establecida. En nuestro caso, dejaremos que P_0 sea generada aleatoriamente por el AG. El proceso que nuestro AG lleva a cabo para producir cada miembro de P_0 es bastante sencillo, pues consiste solamente en escoger un número aleatorio $u \in (0, 1)$ y usarlo en la siguiente función:

$$\psi(u) = (1 - u)C_I + uC_S \tag{1.3}$$

donde C_I y C_S son las restricciones de caja para la variable x. Luego el valor de la función ψ valuada en u corresponderá al individuo $x_0^i \in \mathbb{R}^n$. El proceso anterior es repetido λ veces para generar cada individuo de P_0.

Retomando el ejemplo 1.2, la tabla 1.4 muestra a los individuos de P_0, con su respectivo genotipo y valor de la función de aptitud (fenotipo). Además, la figura 1.2 a) muestra la gráfica de la función 1.2 junto con los individuos de P_0.

Individuo	$\psi(u) = x$ (genotipo)	$f(x)$ (fenotipo)
x_0^1	15.92052	-0.05133
x_0^2	24.55470	-0.05589
x_0^3	22.75560	-0.08246
x_0^4	8.953711	0.34923
x_0^5	17.97670	-0.14624
x_0^6	9.911867	-0.29389

Tabla 1.4: *Individuos de la población inicial P_0 generados aleatoriamente para el ejemplo 1.2.*

1.4. Operadores genéticos

Como se muestra en el pseudocódigo del algoritmo 2, un AG requiere de operadores de selección, cruza y mutación para poder generar una nueva población. Existen distintas clases de operadores genéticos, algunos más complejos que otros, por lo que

en esta sección abordaremos varios tipos, comenzando por los utilizados por nuestro AG.

1.4.1. Selección por torneo

El operador de selección de padres por torneo (que es el utilizado por nuestro AG) es uno de los más flexibles y usados que existen, ya que a diferencia de otros no requiere de un conocimiento global de la población. Éste, consiste únicamente en seleccionar a $k \geq 2$ individuos de la población de forma aleatoria (donde k es el tamaño del torneo), compara sus aptitudes para seleccionar al que tenga el mejor valor en la función objetivo. De este modo, el individuo mejor adaptado es seleccionado de entre los k participantes del torneo. El proceso se repite j veces, con $j \geq 2$, es decir, hasta haber seleccionado tantos padres como el operador de cruza requiera (por lo general 2). Este operador es conceptualmente sencillo y muy fácil de aplicar, la descripción completa es mostrada en el algoritmo 3.

Algoritmo 3: Pseudocódigo del operador de selección por torneo.

```
1 begin
2     índice_de_padres=1;
3     while (índice_de_padres≤ j) do
4         Escoger k individuos de la población de forma aleatoria;
5         Seleccionar al mejor adaptado y denotarlo por P;
6         padres[índice_de_padres]=P;
7         índice_de_padres=índice_de_padres+1;
8     end
9 end
```

Para el AG que usamos en este ejemplo, $k = 2$ y $j = 2$, es decir, la competencia se lleva a cabo entre 2 individuos para escoger un padre, y el procedimiento es llevado a cabo 2 veces para seleccionar 2 padres. Cabe destacar también que, la selección de los 2 competidores es llevada a cabo sin remplazo. Es decir, no es posible seleccionar a un individuo 2 veces para que compita contra el mismo.

Una característica relevante de este operador, es que se puede controlar la *presión de selección*[7] de los individuos al aumentar o disminuir el tamaño del torneo k, ya que entre más alto sea k, mayores serán las probabilidades de seleccionar a los individuos con una adaptación superior al promedio, mientras que valores bajos de k permiten que los individuos menos adaptados también sean seleccionados. Cabe destacar que valores altos de k podrían provocar que la población sea "dominada"

[7] La presión de selección de un individuo, es la probabilidad que tiene éste de ser seleccionado de entre todos los elementos de la población, esta probabilidad dependerá de la aptitud del individuo, así como del mecanismo u operador de selección.

por individuos con una aptitud superior a la del promedio, produciendo así una convergencia prematura [ES03]. Para evitar ésto, en el AG que usamos en ésta sección, la competencia por la selección se lleva a cabo siempre entre 2 individuos, es decir, para $k = 2$.

Una vez que el operador de selección ha escogido 2 padres, digamos x_i^1 y x_i^2, se escoge un numero aleatorio $u \in (0, 1)$ y se compara con p_c para decidir si estos serán cruzados. Si, $u > p_c$ entonces, los padres x_i^1 y x_i^2 no serán cruzados (es decir, no producirán descendencia) y pasarán directamente al siguiente operador. En caso contrario (si $u \leq p_c$) los padres x_i^1 y x_i^2 serán sometidos al *operador de cruza* para producir 2 descendientes x_{i+1}^1 y x_{i+1}^2.

Existen otros operadores de selección de padres que no se basan en un torneo o competencia, tal es el caso del Fitness Proportional Selection (FPS) o el Ranking Selection. Estos operadores no serán explicados en este trabajo pero pueden ser consultados en la referencia [ES03].

1.4.2. Operadores de cruza

El AG que estamos utilizando para este capítulo emplea el operador de cruza binario simulado, o SBX por sus siglas en inglés (*Simulated Binary Crossover*) [DgB95] ya que, según afirman los teóricos, el uso de codificación real en los AG produce un comportamiento más errático y difícil de predecir, en comparación con la codificación binaria [Coe99]. Por lo que antes de explicar el SBX describiremos la cruza binaria basada en un punto, que es precisamente el procedimiento que el SBX emula.

Operador de cruza binario basado en un punto

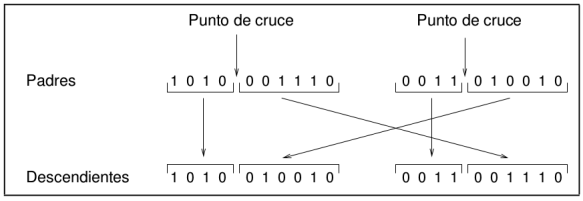


Figura 1.3: *Operador de cruce basado en un punto.*

La cruza binaria basada en un punto consiste básicamente en tomar 2 individuos (previamente escogidos por el operador de selección), digamos 1010001110 y 0011010010, así como un punto de corte aleatorio, digamos 5. Posteriormente cada cadena es dividida en 2 sub-cadenas usando el punto de corte, es decir, 10100 y 01110 para el primer individuo, 00110 y 10010 para el segundo. A la primera sub-cadena generada por el punto de corte le llamamos sub-cadena inicial y a la segunda sub-cadena final. La sub-cadena inicial del padre uno y dos, es transmitida de forma directa al descendiente uno y dos respectivamente, mientras que las sub-cadenas finales de los padres son intercambiadas para conformar las sub-cadenas finales de los

descendientes, formando así dos hijos (1010010010 y 0011001110) con características de ambos padres y con representaciones de igual longitud (en este caso 10 bits). El procedimiento anterior puede ser apreciado de forma más clara en la figura 1.3.

Operador de cruza binario simulado SBX

A diferencia de otros operadores de cruza con representación real, la distribución de probabilidad del SBX es similar a la de un operador de cruza binario. Más aún, el SBX comparte también algunas características muy deseables [DgB95] para la construcción de 2 descendientes a partir de 2 padres;

1. La distancia entre descendientes es proporcional a la distancia entre padres.

2. Los descendientes cercanos a los padres tienen una mayor probabilidad de ser escogidos.

Las características anteriores son de gran importancia para el operador ya que, así como en los organismos biológicos, se espera que los hijos sean de cierto modo parecidos a los padres, esto nos ayuda a preservar y seguir la línea de buenas soluciones.

La primera característica descrita arriba, sugiere que la distancia entre los descendientes deberá estar en función de la distancia entre padres y de un *factor de propagación* β. Dicho factor de propagación está definido como el valor absoluto del cociente de la distancia entre hijos y la distancia entre padres, es decir:

$$\beta = \left| \frac{x_{i+1}^2 - x_{i+1}^1}{x_i^2 - x_i^1} \right| \tag{1.4}$$

Lo cual nos lleva a la segunda característica, una función de densidad $\mathcal{P}$, la cual describe el comportamiento de β y cuya función de distribución de probabilidad favorezca más la elección de descendientes cercanos a los padres (ver figura 1.4), es decir:

$$\mathcal{P}(\beta) = \begin{cases} 0.5(\eta+1)\beta^{\eta}, & \text{si } \beta \leq 1; \\ 0.5(\eta+1)\frac{1}{\beta^{\eta+2}}, & \text{en caso contrario.} \end{cases} \tag{1.5}$$

donde $\eta \in \mathbb{R}^+$ es el índice de distribución.

En la figura 1.4 puede apreciarse que valores altos de η provocan que la distribución de probabilidad favorezca más la selección de hijos cercanos a los padres (en este caso, cercanos al 1), mientras que valores bajos de η mejora las probabilidades de elegir hijos distantes de los padres. Sin embargo, $\eta = 2$ es el que mejor simula el cruce binario basado en un punto [DA94]. De acuerdo con la distribución de probabilidad mostrada en la figura 1.4, ésta favorecerá más a aquellos factores de propagación que se encuentren cercanos a 1. Es decir, a los descendientes que estén cercanos a los padres.

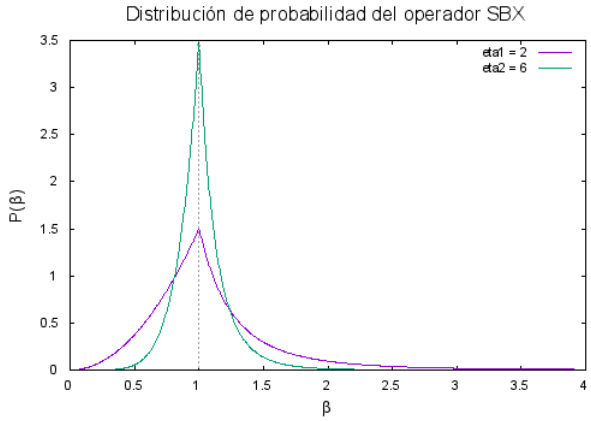


Figura 1.4: *Distribución de probabilidad de* $\mathcal{P}(\beta)$.

El proceso para encontrar el factor de propagación β_q usado en la cruza de cualesquiera dos padres debe ser realizado de forma aleatoria. Para lograr esto, un número $u \in [0, 1)$ es seleccionado al azar, el cual representará el área bajo la curva de probabilidad de la función de densidad desde 0 hasta β_q, es decir:

$$u = \begin{cases} \int_0^{\beta_q} 0.5(\eta+1)\beta^{\eta} d\beta, & \text{si } \beta \leq 1; \\ \int_0^{1} 0.5(\eta+1)\beta^{\eta} d\beta + \int_1^{\beta_q} 0.5(\eta+1)\frac{1}{\beta^{\eta+2}} d\beta, & \text{en caso contrario.} \end{cases} \tag{1.6}$$

luego,

$$u = \begin{cases} \frac{\beta_q^{\eta+1}}{2}, & \text{si } u \leq 1/2; \\ 1 - \frac{\beta_q^{\eta+1}}{2}, & \text{en caso contrario.} \end{cases} \tag{1.7}$$

por último, despejamos β_q tal que

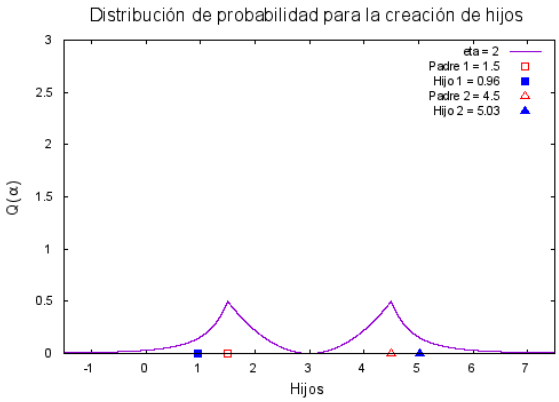
$$\beta_q = \begin{cases} (2u)^{\frac{1}{\eta+1}}, & \text{si } u \leq 1/2; \\ \left(\frac{1}{2(1-u)}\right)^{\frac{1}{\eta+1}}, & \text{en caso contrario.} \end{cases} \tag{1.8}$$

De este modo, el factor de propagación β_q es encontrado, por lo que dados cualesquiera dos padres x_i^1 y x_i^2, los descendientes x_{i+1}^1 y x_{i+1}^2 son calculados utilizando las siguientes ecuaciones:

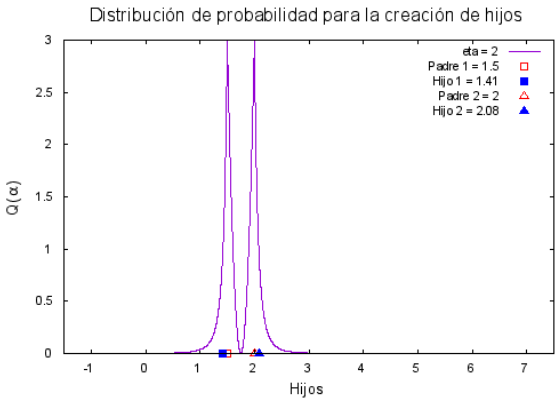
$$x_{i+1}^1 = 0.5[(1+\beta_q)x_i^1 + (1-\beta_q)x_i^2] \tag{1.9}$$

$$x_{i+1}^2 = 0.5[(1-\beta_q)x_i^1 + (1+\beta_q)x_i^2] \tag{1.10}$$

Note que los descendientes x_{i+1}^1 y x_{i+1}^2 son simétricos respecto a los padres x_i^1 y x_i^2, lo cual es una característica premeditada con el fin de no otorgar preferencia por alguno de los padres.



(a) Distribución de $\mathcal{Q}(\alpha)$ con padres lejanos



(b) Distribución de $\mathcal{Q}(\alpha)$ con padres cercanos

Figura 1.5: *Distribución de probabilidad de $\mathcal{Q}(\alpha)$ para 2 padres, con $u = 0.8$ y $\eta = 2$.*

$$(x_{i+1}^2 - x_{i+1}^1) = \beta_q(x_i^2 - x_i^1) \tag{1.11}$$

La ecuación anterior muestra que, sin importar si los padres se encuentran lejanos o no, los hijos se mantendrán a uno distancia proporcional a la de los padres. Más aún, padres lejanos permiten hijos más separados de los padres, mientras que padres cercanos obligan a los hijos a estar cerca de ellos.

Para mostrar esta afirmación, suponga $u = 0.8$ y $\eta = 2$ luego sustituyendo en la ecuación 1.8, tenemos que $\beta_q = 2.5^{1/3}$. Ahora analicemos 2 escenarios, el primero correspondiente a padres lejanos (gráfica 1.5a) y el segundo a padres cercanos (gráfica 1.5b); 1) $x_i^1 = 1.5$ y $x_i^2 = 4.5$, 2) $x_i^1 = 1.5$ y $x_i^2 = 2$. Para el primer caso, se tiene que el hijo $x_{i+1}^1 = 0.96$ se encuentra más alejado que el padre $x_i^1 = 1.5$ al ser comparado con el segundo caso, donde el hijo $x_{i+1}^1 = 1.41$ está mucho más cercano al padre $x_i^1 = 1.5$. Las figuras 1.5 a y 1.5 b ilustran la proporción guardada entre padres e hijos mostrada en la ecuación 1.11 y el ejemplo anterior.

Las gráficas del grupo de figuras 1.5, fueron generadas con el fin de ilustrar la proporción y simetría que existe entre los padres y sus respectivos descendientes. Sin embargo, la función de densidad usada para generar dichas gráficas (la cual denotaremos por $\mathcal{Q}$) discrepa de la mostrada en la ecuación 1.5. La construcción de la función de densidad $\mathcal{Q}$ es explicada a continuación.

Suponga que $x_i^1 \leq x_i^2$. Para construir $\mathcal{Q}$ sera necesario llevar a cabo tres tareas; 1) Mover la función de distribución $\mathcal{P}$ una cantidad $a = x_i^2 - (x_i^2 - x_i^1)/2 \geq 0$, con el fin de que la función de densidad $\mathcal{Q}$ se encuentre centrada en el punto medio entre los dos padres $x_i^1 \leq x_i^2$, es decir, tenga media $\mu = x_i^2 - (x_i^1 - x_i^2)/2$. 2) Multiplicar la función de densidad $\mathcal{P}$ por una cantidad $b \in (0, 1)$ con la finalidad de reducir el área bajo la curva de probabilidad de 1 a 1/2. Esto último se debe a que, la función de densidad $\mathcal{Q}$ tiene dos "campanas", una sobre cada padre, por lo cual es necesario que el área bajo cada una sea 1/2 y de éste modo, al calcular el área total bajo la curva

de probabilidad de $\mathcal{Q}$, ésta sea 1. 3) Después crear la variable aleatoria continua $\alpha = a + b\beta$ por medio de los dos pasos anteriores, solo queda construir la campana que estará sobre el padre x_i^1. Para llevar a cabo ésto, bastara con "duplicar" $\mathcal{P}(\alpha)$ haciendo una reflexión horizontal a la izquierda.

Primer y segundo paso, construir la campana derecha centrada en x_i^2. Dada la variable aleatoria continua β con función de densidad $\mathcal{P}(\beta)$, debemos encontrar la función de densidad $g(\alpha)$ correspondiente a la variable aleatoria continua $\alpha = a+b\beta$, lo que nos lleva al siguiente teorema([PGH71], sec. 5.2.1, Teorema 1).

Teorema 1.1. *Sea X una variable aleatoria continua con función de densidad f, además sean a y b constantes tal que $b > 0$. Entonces, la función de densidad de la variable aleatoria continua $Y = a + bX$ está dada por*

$$g(y) = \frac{1}{b} f\left(\frac{y-a}{b}\right), \qquad con \ -\infty < y < \infty. \tag{1.12}$$

Del teorema anterior y la ecuación 1.5 se sigue que la función de densidad $g(\alpha)$ es

$$g(\alpha) = \begin{cases} \frac{\eta+1}{2b}\left(\frac{\alpha-a}{b}\right)^{\eta}, & \text{si } \ a < \alpha \leq x_i^2; \\ \frac{\eta+1}{2b}\left(\frac{b}{\alpha-a}\right)^{\eta+2}, & \text{si } \ x_i^2 < \alpha < \infty. \end{cases} \tag{1.13}$$

donde $a = x_i^2 - (x_i^2 - x_i^1)/2$.

El siguiente paso es encontrar el valor de b para cada caso de la función 1.13. Puesto que los demás parámetros son conocidos, bastara con integrar cada caso e igualar con $1/4$[8]. Veamos el primer caso:

$$\frac{1}{4} = \int_a^{x_i^2} \frac{\eta+1}{2b}\left(\frac{\alpha-a}{b}\right)^{\eta} d\alpha = \frac{1}{2}\left(\frac{x_i^2 - x_i^1}{2b}\right)^{\eta+1} \tag{1.14}$$

despejando tenemos que $b = \frac{x_i^2 - x_i^1}{2^{\eta/(\eta+1)}}$.
Del mismo modo para el segundo caso de la ecuación 1.13:

$$\frac{1}{4} = \int_{x_i^2}^{\infty} \frac{\eta+1}{2b}\left(\frac{b}{\alpha-a}\right)^{\eta+2} d\alpha = \frac{1}{2}\left(\frac{2b}{x_i^2 - x_i^1}\right)^{\eta+1} \tag{1.15}$$

despejando tenemos que $b = \frac{x_i^2 - x_i^1}{2^{\frac{\eta+2}{\eta+1}}}$. luego, note que:

$$a = \frac{x_i^2 - (x_i^2 - x_i^1)}{2} = \frac{x_i^2 + x_i^1}{2}$$

por lo que la ecuación 1.13 queda como:

[8]Ya que recuerde que el área total bajo la curva de la función $g(\alpha)$ debe ser $1/2$. Además, el área bajo la curva de cada caso de la función de densidad $\mathcal{P}(\alpha)$ es $1/2$ [DA94].

$$g(\alpha) = \begin{cases} \frac{(\eta+1)2^{\eta-1}}{(x_i^2-x_i^1)^{\eta+1}}\left(\alpha - \frac{x_i^2+x_i^1}{2}\right)^{\eta}, & \text{si } \frac{x_i^2+x_i^1}{2} \leq \alpha \leq x_i^2; \\ \frac{(\eta+1)(x_i^2-x_i^1)^{\eta+1}}{2^{\eta+3}}\left(\alpha - \frac{x_i^2+x_i^1}{2}\right)^{-(\eta+2)}, & \text{si } x_i^2 < \alpha < \infty. \end{cases} \tag{1.16}$$

Por último, solo queda realizar la reflexión de la función 1.16 a la izquierda para generar la campana centrada en x_i^1, es decir:

$$g(-\alpha) = \begin{cases} \frac{(\eta+1)(x_i^2-x_i^1)^{\eta+1}}{2^{\eta+3}}\left(\frac{x_i^2+x_i^1}{2} - \alpha\right)^{-(\eta+2)}, & \text{si } -\infty < \alpha < x_i^1; \\ \frac{(\eta+1)2^{\eta-1}}{(x_i^2-x_i^1)^{\eta+1}}\left(\frac{x_i^2+x_i^1}{2} - \alpha\right)^{\eta}, & \text{si } x_i^1 \leq \alpha \leq \frac{x_i^2+x_i^1}{2}. \end{cases} \tag{1.17}$$

luego, la función de densidad $\mathcal{Q}(\alpha)$ para 2 padres x_i^1 y x_i^2 es:

$$\mathcal{Q}(\beta) = \begin{cases} g(-\alpha), & \text{si } -\infty < \alpha < \frac{x_i^2+x_i^1}{2}; \\ g(\alpha), & \text{si } \frac{x_i^2+x_i^1}{2} \leq \alpha < \infty. \end{cases} \tag{1.18}$$

Una vez que el operador de cruza ha producido 2 descendientes x_{i+1}^1 y x_{i+1}^2, es necesario decidir de forma individual si serán sometidos al operador mutación. Para ello, se escoge un numero aleatorio $u \in (0,1)$ y se compara con p_m. Si, $u > p_m$ entonces, el descendiente x_{i+1}^1 no será mutado y pasará directamente formar parte de P_{i+1}. En caso contrario (si $u \leq p_m$), el descendiente x_{i+1}^1 será sometidos al operador de mutación (explicado a continuación) para luego ser integrado a P_{i+1}. El mismo procedimiento es realizado para el descendiente x_{i+1}^2.

1.4.3. Operador de mutación

El operador de mutación es bastante parecido al operador de cruza SBX, pero a diferencia de éste, trabaja individualmente sobre cada descendiente x_{i+1}^1 y x_{i+1}^2. Es decir, para x_{i+1}^1 (respectivamente x_{i+1}^2) el operador de mutación cambia su valor por uno que este en una vecindad de x_{i+1}^1, siempre y cuando $u \leq p_m$[9], de lo contrario x_{i+1}^1 no es mutado y pasan a formar parte de la siguiente generación P_{i+1}. La Vecindad de x_{i+1}^1 es escogida usando una distribución de probabilidad polinomial[DG96] con $\mu = x_{i+1}^1$ y desviación estándar σ, la cual, está en función del índice de distribución de mutación $\nu \in \mathbb{N} \cup \{0\}$.

Para generar la mutación c es necesario definir un factor de perturbación δ:

[9]Cabe destacar que cada vez que un número aleatorio $u \in (0,1)$ es utilizado, éste es distinto que cualquier otro antes mencionado, es decir, el número aleatorio u comparado con p_c es distinto al utilizado al comparar p_m, aunque se trate de la misma tupla de padres.

$$\delta = \frac{c - x_{i+1}^1}{C_S - C_I} \tag{1.19}$$

donde c es el valor de x_{i+1}^1 mutado. Similarmente al operador de cruza SBX, el valor de la mutación c es escogido usando la distribución de probabilidad mostrada en la figura 1.6, cuya función de densidad que depende de δ:

$$\mathcal{P}(\delta) = 0.5(\nu + 1)(1 - |\delta|)^\nu \qquad \delta \in (-1, 1) \tag{1.20}$$

Para ello, generamos un número aleatorio $u \in (0, 1)$ y encontramos el valor δ_q haciendo el área bajo la curva de la función de densidad $\mathcal{P}(\delta)$ de -1 a δ_q igual a u, tal que:

$$\delta_q = \begin{cases} (2u)^{\frac{1}{\nu+1}}, & \text{si } \ u < 1/2; \\ 1 - [2(1-u)]^{\frac{1}{\nu+1}}, & \text{en cc.} \end{cases} \tag{1.21}$$

Por último, el valor de la mutación c de x_{i+1}^1 es calculado como sigue:

$$c = x_{i+1}^1 + \delta_q(C_S - C_I) \tag{1.22}$$

La distribución de probabilidad $\mathcal{P}(\delta)$ es válida solo para $\delta \in (-1, 1)$ y es mostrada en la figura 1.6 con valores distintos de ν. Note que, así como en el operador de cruza SBX, el parámetro ν determina el ancho y alto de la distribución, haciendo que valores altos de ν incrementen la probabilidad de escoger mutaciones c cerca del individuo x_{i+1}^1, y valores bajos de ν dan más probabilidades de ser escogidas las mutaciones más lejanas de x_{i+1}^1 (ver figura 1.6). Por ejemplo, $\nu = 0$ produce una distribución de probabilidad uniforme, es decir, todas las mutaciones c de x_{i+1}^1 tienen las mismas probabilidades de ser seleccionadas, incluso las más lejanas, mientras que $\nu = 100$ hace casi imposible la elección de mutaciones lejanas a x_{i+1}^1.

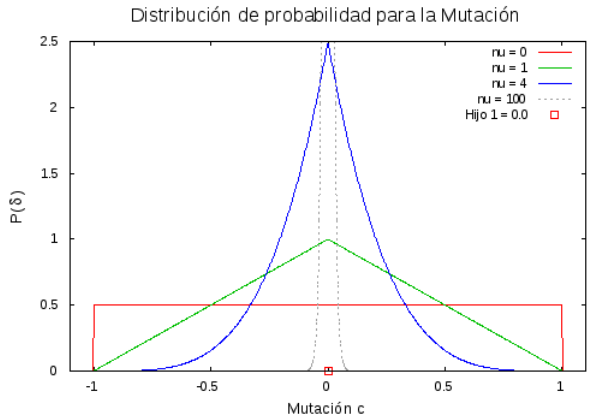


Figura 1.6: *Distribución de probabilidad para la mutación del individuo $x = 0$.*

La mutación es el último operador que modifica el cromosoma (genotipo) de los individuos en la población. Con el fin de mantener la notación y no complicarla, hagamos:

$$x_{i+1}^1 = c \tag{1.23}$$

Como se indicó al inicio de la explicación del operador de mutación, se aplica el mismo procedimiento al segundo descendiente de x_i^1 y x_i^2, es decir, a x_{i+1}^2.

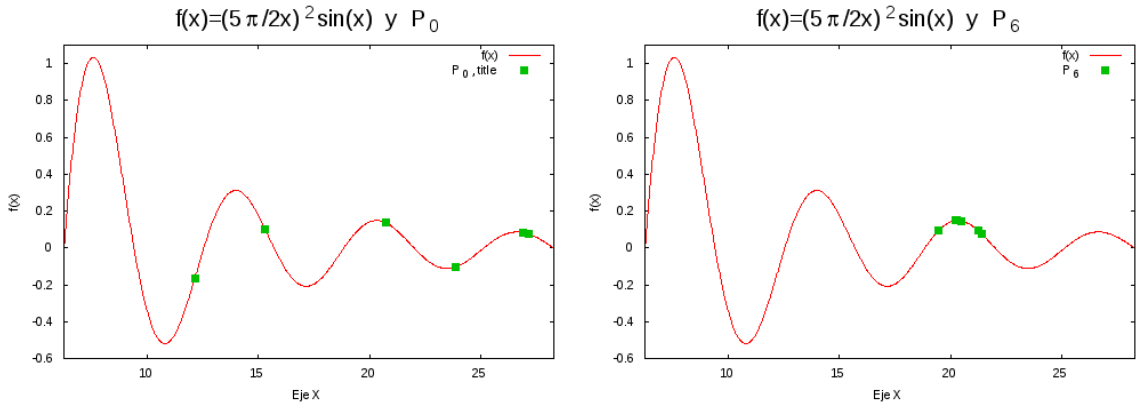
Retomando el ejemplo 1.2, la tabla 1.5 muestra a P_1 generada a partir de P_0 (mostrada en la tabla 1.4) y los operadores de selección, cruza y mutación.

Individuo	Padres seleccionados	x_1^i (cruzado)	x_1^i (mutado)	$f(x_1^i)$ (aptitud)
x_1^1	$x_0^2 = 24.55480$	24.55480	24.40446	-0.06894
x_1^2	$x_0^1 = 15.92046$	15.92046	16.39832	-0.14608
x_1^3	$x_0^4 = 8.95368$	7.25610	7.25610	0.96835
x_1^4	$x_0^2 = 24.55480$	26.25238	26.25238	0.08055
x_1^5	$x_0^4 = 8.95368$	8.86338	8.86338	0.41802
x_1^6	$x_0^1 = 15.92046$	16.01076	15.51458	0.04925

Tabla 1.5: *Individuos de P_1 generados para el ejemplo 1.2.*

La primera columna muestra a los individuos de P_1, la segunda muestra la tupla de padres correspondiente a cada tupla de hijos, note que los padres $x_0^1,\ x_0^2, x_0^4 \in P_0$ fueron escogidos por el operador de selección por torneo para ser padres (en tres torneos individuales). Más aún, cada uno fue seleccionado dos veces, mientras que x_0^3, x_0^5 y x_0^6 no fueron escogidos ni una vez. La tercera columna muestra a los individuos de P_1 después de aplicar el operador de cruza a los individuos de P_0 seleccionados. Note que, los padres x_0^2 y x_0^1 de los descendientes x_1^1 y x_1^2 no fueron cruzados (debido a que la probabilidad de cruza no fue suficientemente alta, $p_m = 0.8$), por lo que pasaron directamente a formar parte de P_1 (no sin antes pasar por el operador de mutación), mientras que los demás padres seleccionados de P_0 si fueron cruzados. La cuarta columna muestra a los individuos de P_1 después de aplicar el *operador de mutación* a los individuos resultantes del operador de cruza, note que los individuos x_1^1, x_1^2 y x_1^6 fueron mutados, más aún, 2 de ellos presentan un decremento en su valor, mientras que el otro (x_1^2) un aumento. Por último, la quinta columna muestra la aptitud de cada individuo, es decir, la función objetivo (ecuación 1.2) valuada en el fenotipo de cada individuo de P_1 (x_1^i mutado). Los individuos de P_2 pueden ser apreciados en la figura 1.2 b.

Otra cosa importante a notar en la tabla 1.5 es la similitud entre los valores de las columnas 2 y 3, es decir, los descendientes son cercanos a los padres, salvo el primer y segundo individuo, ya que los padres no fueron cruzados. Del mismo modo para



(a) $f(x)$ y P_0, con $\gamma = 0.123$ y $\nu = 15$ **(b)** $f(x)$ y P_6, con $\gamma = 0.123$ y $\nu = 15$

Figura 1.7: *Evolución de P_0 a un máximo local, con $\gamma = 0.123$ y $\nu = 15$, para el ejemplo 1.2.*

las columnas 3 y 4, las mutaciones (para los individuos a los que se les aplicó) son aún más cercanas a los individuos antes de aplicarles el operador de mutación. Esto resalta el valor de nuestros operadores de cruza (SBX) y mutación, ya que, dicho coloquialmente, estos garantizan que "el fruto no caiga muy lejos del árbol".

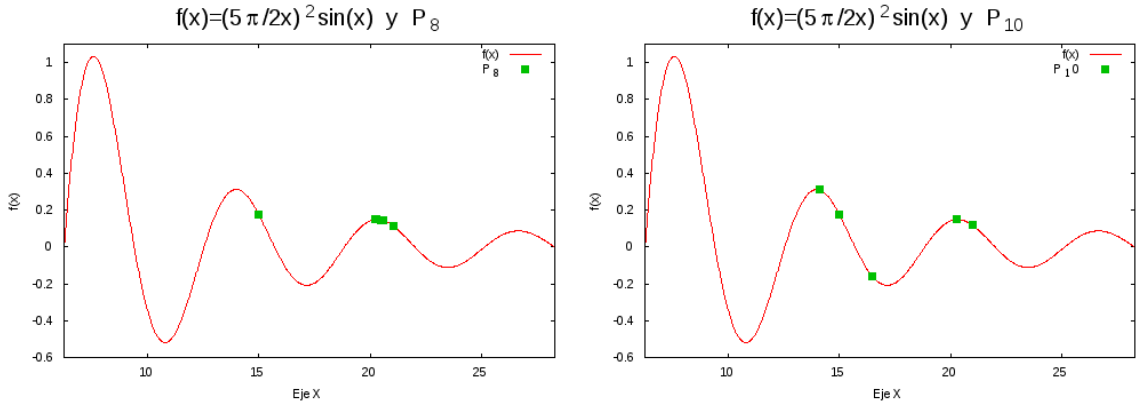
El argumento anterior es aún más notorio si calculamos la distancia promedio entre descendientes (sin mutar) y padres, la cual es 0.9, y la comparamos con la distancia promedio entre descendientes antes y después de ser mutados, la cual es 0.4, (ya que en el ejemplo 1.2 $\eta = 2 < \nu = 100$)[10].

El conjunto de figuras 1.7 tiene como objetivo mostrar la importancia del operador de mutación en los AGs. Ya que como puede apreciarse en la figura 1.7b, en caso de que el AG llegase a concentrar a los individuos de la población alrededor de un optimo local, el operador de mutación es capaz de hacer que el AG siga explorando otras zonas de la región factible en busca de mejores soluciones. En caso de que esto suceda, el AG eventualmente hará que la población se mueva a la región donde el valor de la función objetivo sea mas alta (o baja para un problema de minimización), tal como puede apreciarse en el grupo de figuras 1.8.

Como puede observarse en la figura 1.6, el valor del parámetro ν determinara la probabilidad de que mutaciones mas agresivas y alejadas del individuo original sean elegidas. Por lo que la elección de valeres adecuados de ν (en este caso, no demasiado grandes), determinara la facilidad y rapidez con la que el AG sera capaz de salir de óptimos locales en donde se encuentre estancado, y así seguir buscando sobre la región factible, hasta eventualmente encontrar el óptimo global.

La convergencia de la población hacia el óptimo global está determinada por todos

[10]Esto se debe a la distribución de probabilidad de los operadores de cruza y mutación, particularmente a los parámetros η y ν respectivamente, ya que, como se explicó anteriormente, valores altos de dichos parámetros favorece la elección de individuos cercanos a sus padres, y en el caso de la mutación, a los individuos cercanos al descendiente no mutado



(a) $f(x)$ y P_8, con $\gamma = 0.123$ y $\nu = 15$

(b) $f(x)$ y P_{10}, con $\gamma = 0.123$ y $\nu = 15$

(c) $f(x)$ y P_{22}, con $\gamma = 0.123$ y $\nu = 15$

(d) $f(x)$ y P_{35}, con $\gamma = 0.123$ y $\nu = 15$

Figura 1.8: *Evolución de la población desde un óptimo local hasta alcanzar el global.*

los parámetros y no solo por ν, el objetivo de los grupos de figuras 1.7 y 1.8 es mostrar que variaciones en los parámetros, aunque sea solo en uno, pueden determinar el éxito o fracaso del algoritmo, al mismo tiempo que muestra la flexibilidad para adaptarse a distintos tipos de problemas. En las siguientes secciones analizaremos problemas mucho más complejos que el ejemplo 1.2, y aprovecharemos para mostrar la importancia y sensibilidad de los demás parámetros.

1.4.4. Selección de supervivientes

El algoritmo 2 muestra que después de llevar a cabo la selección de padres, cruza y mutación, es necesario extender la población actual, digamos P_i, de λ a 2λ individuos agregando a los descendientes producto de dichos operadores. Posteriormente la población extendida es sometida al operador de selección de supervivientes que escogerá a los individuos que conformaran la siguiente generación P_{i+1}.

Como pudo apreciarse a lo largo de esta sección, el AG que estamos usando no realiza la extensión de la población ni la selección de supervivientes, en lugar de ello utiliza a los individuos de P_i escogidos por el operador de selección de padres (selección por torneo) para conformar la siguiente generación P_{i+1} (no sin antes haberlos sometido a los operadores de cruza y mutación), haciendo así más eficiente el algoritmo.

Sin embargo, existen ciertas ventajas (y desventajas, dependiendo del problema con el que se trata) al realizar un proceso de selección más exhaustivo, por lo que vale la pena explican algunos operadores de selección de supervivientes. En muchos textos llaman a este proceso remplazo, por lo que en lo subsecuente también nos referiremos al proceso de selección de supervivientes del mismo modo, ya que además es un término más práctico.

Las principales diferencias entre los operadores de remplazo radican en la manera en la que estos escogen a los individuos de entre la población extendida, por ejemplo, basándose en la "edad" de los individuos o su aptitud.

Remplazo basado en la aptitud

Un gran número de estrategias de remplazo han sido propuestas para seleccionar a los λ individuos de entre los 2λ que conformaran la siguiente generación. Muchos de ellos (como el nuestro) utilizan la edad como criterio para asegurar que todos los descendientes pasen a la siguiente generación.

En el esquema de Remplazo de los peores, los λ miembros menos adaptados de la población extendida son extraídos, de tal modo que los λ restantes pasan a formar parte de la siguiente generación. Sin embargo, esto puede provocar un incremento muy rápido en la aptitud promedio de la población de individuos, y posiblemente la convergencia prematura del algoritmo, debido a que la población es "dominada" por los individuos mejor adaptados. Por ello algunas veces, este esquema es mezclado con otros y/o se le aplican políticas restrictivas con el fin de evitar la convergencia

prematura del AG[11]. El esquema *Elitista* es menos agresivo que el de remplazo de los peores, y se enfoca en mantener al mejor individuo de cada generación. Básicamente si el individuo mejor adaptado de la población actual no fue seleccionado para formar parte de la siguiente generación y ninguno de los descendientes tiene una aptitud igual o mejor que él, entonces éste pasa a la siguiente generación en lugar de algún descendiente. Comúnmente éste esquema es usado en conjunto con otros, ya sea basados en la edad o en la aptitud individual.

[11] [ES03], Sección 3.8.2

1.5. Resultados numéricos

En esta sección analizaremos algunas funciones de prueba bien conocidas con el fin mostrar las ventajas de los AG sobre los métodos convencionales tales como Newton, Pendiente Máxima (Steepest Descent), etc.[12]. Antes de seguir adelante definiremos el criterio con el cual mediremos la calidad de las soluciones que aproximemos.

Definición 1.3. *Dado un valor x y una de sus aproximaciones x_{aprox}, definimos el* ***error absoluto*** *ϵ como*

$$\epsilon = |x - x_{aprox}| \tag{1.24}$$

Observación. *Nos referiremos al óptimo global, de un problema de optimización dado, como x^*.*

1.5.1. Función de Beale

Ejemplo 1.3. *Considere el problema de optimización:*

$$\begin{aligned} \textit{Minimizar} \quad & f(x) = A(x) + B(x) + C(x), \ \textit{donde}; \\ & A(x) = (1.5 - x_1 + x_1x_2)^2; \\ & B(x) = (2.25 - x_1 + x_1x_2^2)^2; \\ & C(x) = (2.625 - x_1 + x_1x_2^3)^2; \\ \textit{Sujeto a} \quad & -4.5 \leq x_1, x_2 \leq 4.5; \end{aligned}$$

con mínimo global en $x^* = (3, 0.5)$ y $f(x^*) = 0$.

La función de Beale (función $f(x)$ del ejemplo 1.3) tiene ciertas características que la hacen útil como función de prueba. Ya que, como puede apreciarse en la figura 1.9, la pendiente en el "valle" de la gráfica es muy pequeña, lo que les dificulta a los métodos convencionales como el de Pendiente Máxima encontrar el mínimo. Métodos como el de Pendiente Máxima tienen un rendimiento pobre en esta clase de problemas, debido a que la pendiente al rededor del óptimo global x^* es casi cero.

Utilizando los parámetros mostrados en la tabla 1.6, el AG que usamos es capaz de aproximar el óptimo global x^* en tan solo 28 generaciones[13].

Además de ser el mejor adaptado en la generación 28, el individuo $x_{28}^{32} = (2.999434, 0.500073)$ fue el mejor adaptado de todos los individuos en las 50 generaciones, presentado el error absoluto más bajo respecto a la aptitud $\epsilon =$

[12] Los cuales requieren de cálculos de derivadas (e incluso segundas derivadas) y son de naturaleza local.

[13] Notará que en los parámetros se especifican 50 generaciones (o ejecuciones del AG), cuando el óptimo global fue aproximado en 28. Esto se debe a que, puesto que es una Heurística, la cantidad de generaciones necesarias para encontrar el óptimo global se desconoce, por lo que es necesario darle cierta "tolerancia" al algoritmo para encontrar el óptimo.

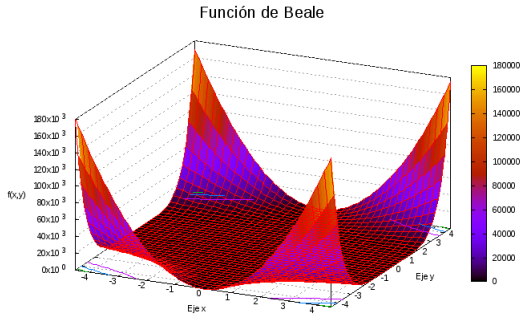


Figura 1.9: *Gráfica de la función de Beale.*

Parámetro	G	λ	P_c	P_m	γ	η	ν
Valor	50	100	0.9	0.3	0.123	2	100

Tabla 1.6: *Parámetros usados en la función de Beale.*

$|f(x^*) - f(x_{28}^{32})| = 0.000001$. La figura 1.10 a muestra la región factible junto con P_0, contrastando con la figura 1.10 b que muestra la región factible junto con P_{28}. La figura 1.11 muestra como disminuye el error absoluto con respecto a la aptitud del individuo mejor adaptado en cada generación. Note que la gráfica no es monótona decreciente, esto se debe a que el algoritmo que usamos no es elitista (Sección 1.4.4), por lo que no necesariamente el individuo mejor adaptado de la generación actual pasa a formar parte de la siguiente generación.

Puesto que muchos de los operadores en los AG son de naturaleza estocástica (en el caso del nuestro todos), podría pensarse que la convergencia del mismo es una cuestión de "suerte". Por ello, hemos decidido ejecutar el algoritmo 30 veces, variando la semilla aleatoria γ en cada ejecución y mostrar en una gráfica de caja y bigote (figura 1.12) el error absoluto ϵ del individuo mejor adaptado de cada ejecución y generación. De la figura 1.12 podemos notar que la convergencia del AG no depende del valor de la semilla aleatoria γ.

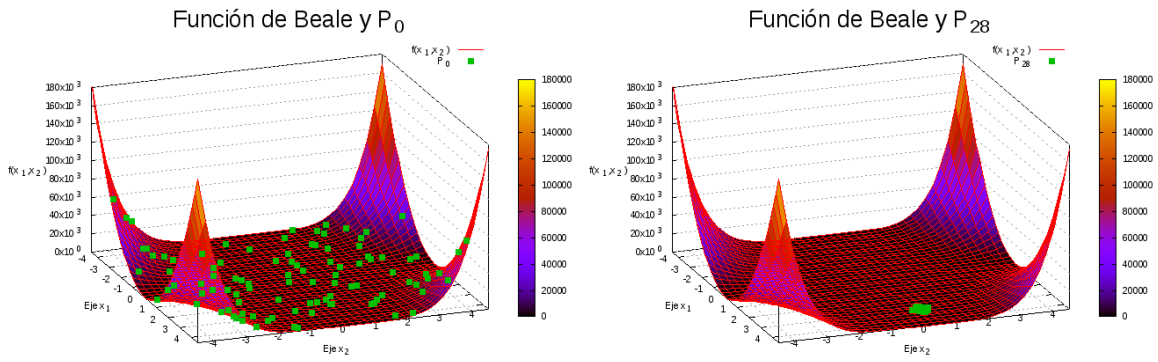
1.5.2. Función de Rastrigin

Ejemplo 1.4. *Considere el problema de optimización:*

$$\begin{aligned} \textit{Minimizar} \quad & f(x) = 10n + \sum_{i=1}^{n} [x_i^2 - 10cos(2\pi x_i)]; \\ \textit{Sujeto a} \quad & -5.12 \leq x_i \leq 5.12, \ \textit{para } i = 1, ..., n; \end{aligned}$$

con $x \in \mathbb{R}^n$, *mínimo global en* $x^* = 0$ *y* $f(x^*) = 0$.

Para el ejemplo 1.4 suponga $n = 2$. La función de Rastrigin (función $f(x)$ del ejemplo 1.4) es otro excelente ejemplo para analizar, ya que como puede observarse



(a) Población inicial. **(b)** Población después de 28 generaciones.

Figura 1.10: *Evolución de la población inicial hacia el óptimo global.*

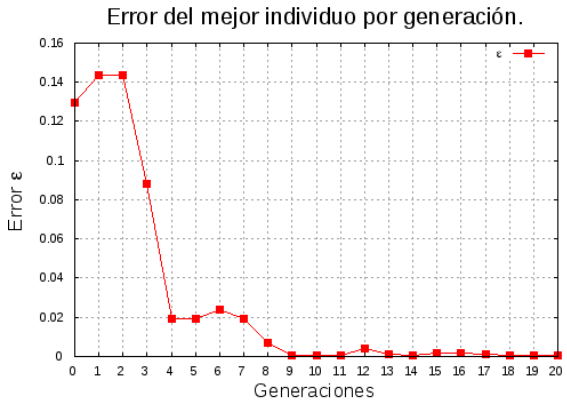


Figura 1.11: *Error absoluto con respecto a la aptitud ϵ del mejor individuo en cada generación.*

en la figura 1.13 esta tiene, coloquialmente hablando, la forma de un cartón de huevo hundido por en medio. En otras palabras, la función de Rastrigin tiene muchos óptimos locales y solo un óptimo global, por lo que los métodos convencionales fácilmente quedan atrapados en alguno de estos óptimos locales.

Debido a lo explicado en el párrafo anterior, es necesario ajustar un poco los parámetros de nuestro AG para permitirle salir de los óptimos locales cada vez que caiga en alguno de ellos. Intuitivamente podríamos disminuir el valor de nuestro parámetro ν (como se hizo en la 2da parte del ejemplo 1.2, grupo de figuras 1.8) para que nuestro AG produzca mutaciones más agresivas, y así permitirle salir de óptimos locales y seguir con su búsqueda del óptimo global. Sin embargo, en esta ocasión además de disminuir un poco el parámetro ν, también disminuiremos en una unidad el parámetro η, con el fin de permitirle a nuestro operador de cruza SBX escoger descendientes un poco más lejanos de los padres.

Utilizando los parámetros especificados en la tabla 1.7, el algoritmo produjo su

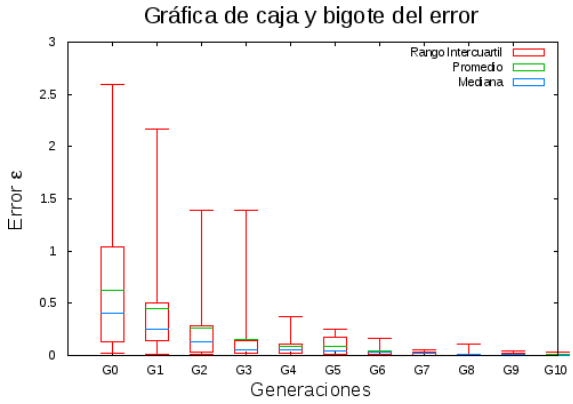


Figura 1.12: *Cada caja representa la distribución del error de los individuos mejor adaptados por generación en las 30 ejecuciones.*

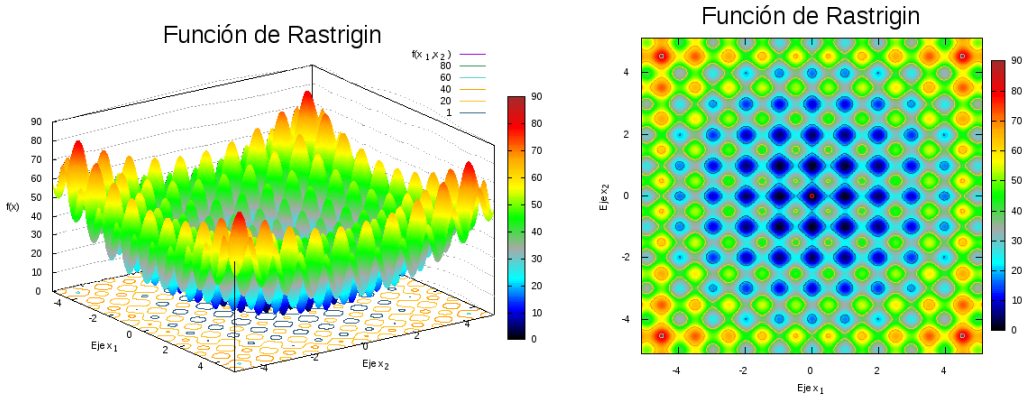
Parámetro	G	λ	P_c	P_m	γ	η	ν
Valor	100	100	0.9	0.3	0.123	1	90

Tabla 1.7: *Parámetros usados en la función de Rastrigin.*

mejor aproximación al óptimo global $x^* = 0$ en la generación 99 con el individuo 23, es decir $x_{99}^{23} = (0.001101, -0.001406)$ (figura 1.14), con un error absoluto respecto a la aptitud $\epsilon = |f(x^*) - f(x_{99}^{23})| = 0.000633$.

Note que a diferencia de la función de Beale, duplicamos el número de generaciones y disminuimos los parámetros ν y η. Debido a que la función de Rastrigin tiene muchos óptimos locales, el algoritmo debe salir de cada uno de ellos en los que cae, lo cual incrementa el número de iteraciones necesarias para poder encontrar el óptimo global. Es por ello que los parámetros ν y η son también más bajos, ya que esto les permite a los individuos de la población dar "saltos" más grandes al mutar o producir hijos más lejanos a los padres y así salir de los óptimos locales. Sin embargo, esto dificulta seguir la línea de buenas soluciones, ya que si inicialmente algún individuo estaba cerca de x^* y este fue estocásticamente seleccionado para ser mutado, o su descendiente fue escogido demasiado lejos, entonces probablemente el individuo o descendiente se aleje de x^*, como puede verse en el error absoluto con respecto al óptimo del individuo mejor adaptado $\epsilon = 0.000633$ que fue mayor a su equivalente en la función de Beale. Pero estas noticias no son necesariamente malas, ya que disminuir estos parámetros podría ayudarnos a conocer mejor la región factible de nuestro problema de optimización, realizando una búsqueda preliminar del óptimo global, es decir, a partir de los resultados obtenidos podemos inferir que x^* se encuentra en una vecindad de x_{99}^{23}.

Otra posible alternativa para atacar la función de Rastrigin podría ser aumentar el tamaño de la población λ (manteniendo $\nu = 100$ y $\eta = 2$) con el fin de "permear" más exhaustivamente la región factible. Esta afirmación, así como las mencionadas en el párrafo anterior, son verificadas al ejecutar nuestro AG una vez más con $\lambda = 150$,



(a) Función de Rastrigin.

(b) Gráfica de niveles de la función de Rastrigin.

Figura 1.13: *Gráficas de la función de Rastrigin.*

$\nu = 100$ y $\eta = 2$ (dejando los demás parámetros iguales), ya que con estos parámetros el mejor individuo fue encontrado en la generación 31 ($x_{31}^{15} = (0.000001, -0.000348)$) teniendo un error absoluto con respecto al óptimo global $\epsilon = 0.000024$ que es una mejor aproximación que la hallada con $\lambda = 100$, $\nu = 90$ y $\eta = 1$ (sin mencionar que fue encontrada en menos iteraciones), figuras 1.15.

Mostramos ambas perspectivas puesto que no siempre es posible aumentar el tamaño de la población, ya sea por una limitante en el equipo de cómputo que usamos o porque esto podría aumentar considerablemente el costo y/o tiempo de convergencia del algoritmo. Sin importar cuál sea el caso, lo anterior muestra la flexibilidad de los AG.

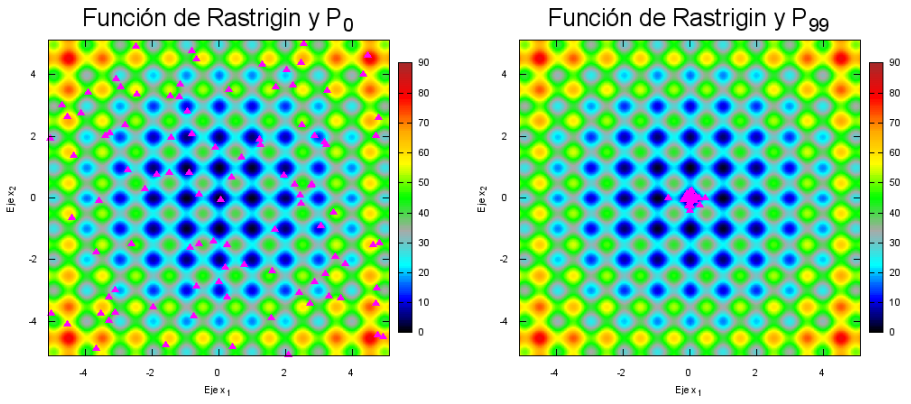
Para finalizar este ejemplo y exponer una vez más que la convergencia de la población es consistente, a pesar de la naturaleza estocástica de los operadores utilizados, en la figura 1.16 mostramos la gráfica de caja y bigote formada con el error absoluto con respecto a x^* de los mejores individuos por generación y ejecución (de 30 ejecuciones del AG) variando la semilla aleatoria γ en cada ejecución y dejando los demás parámetros fijos ($n = 100$, $\lambda = 150$, $P_c = 0.9$, $P_m = 0.3$, $\eta = 2$, $\nu = 100$).

1.5.3. Función de Rosenbrock

Ejemplo 1.5. *Considere el problema de optimización:*

$$\begin{aligned} \textit{Minimizar} \quad & f(x) = \sum_{i=1}^{n-1} [100(x_i^2 - x_{i+1})^2 + (x_i - 1)^2]; \\ \textit{Sujeto a} \quad & -5 \leq x_i \leq 10, \ \textit{para } i = 1, ..., n; \end{aligned}$$

con $x \in \mathbb{R}^n$, *mínimo global en* $x^* = 1$ *y* $f(x^*) = 0$.



(a) Población inicial.

(b) Población inicial después de 99 de generaciones.

Figura 1.14: *Gráficas de la función de Rastrigin, con población inicial y la no. 99 donde se encontró la mejor aproximación de x^*.*

Para el primer análisis del ejemplo 1.5 suponga $n = 2$. La función de Rosenbrock (función $f(x)$ del ejemplo 1.5) es una función no-convexa usada para medir el desempeño de algoritmos de optimización diseñada por Howard H. Rosenbrock en 1960. El mínimo global se encuentra dentro de un valle parabólico, largo y estrecho (figura 1.17), por lo que usualmente también es llamada función del valle. Dicho valle es bastante fácil de encontrar, sin embargo, hallar el mínimo dentro de este no es trivial.

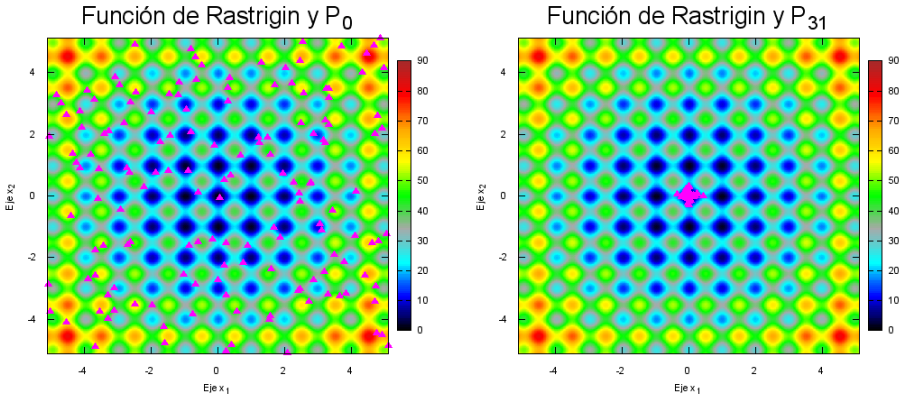
Usando los parámetros mostrados en la tabla 1.8, nuestro AG encontró la mejor aproximación del mínimo global en la generación 347 con el individuo $x_{347}^{69} = (1.00055,\ 1.00092)$, con un error absoluto $\epsilon = 0.000004$ en menos de 3 décimas de segundo.

Parámetro	G	λ	P_c	P_m	γ	η	ν
Valor	500	100	0.9	0.3	0.123	25	100

Tabla 1.8: *Parámetros usados en la función de Rosenbrock.*

Sin embargo, en la mayoría de los problemas de optimización con aplicaciones reales el número de variables con las que se trabaja es mayor que 2, y en muchos casos son incluso miles. A medida que n aumenta la complejidad para encontrar el óptimo también lo hace, ya que el costo del algoritmo para encontrarlo puede crecer desmesuradamente, provocando que los métodos convencionales sean incapaces de encontrar el óptimo en un tiempo aceptable.

Debido a la naturaleza heurística de los AG estos son capaces de encontrar soluciones aceptables en tiempos aceptables, para problemas donde los métodos convencionales fallan. Para ilustrar esto extenderemos el espacio de la función de Rosenbrock



(a) Función de Rastrigin y P_0, con $\lambda = 150$, $\nu = 100$ y $\eta = 2$.

(b) Función de Rastrigin y P_{31}, con $\lambda = 150$, $\nu = 100$ y $\eta = 2$.

Figura 1.15: *Gráficas de la función de Rastrigin para $m = 2$, $\lambda = 150$, $\nu = 100$ y $\eta = 2$; con P_0 y P_{31} donde se encontró la mejor aproximación de x^*.*

a $n = 10$, mostrando que con un simple ajuste de parámetros podemos robustecer nuestro AG y hacerlo capaz de resolver problemas más complejos. Para ello ejecutaremos nuestro AG 30 veces con los parámetros mostrados en la tabla 1.9, variando la semilla aleatoria γ en cada uno de ellos.

Parámetro	G	λ	P_c	P_m	γ	η	ν
Valor	150,000	100	0.9	0.3	0.123	25	100

Tabla 1.9: *Parámetros usados en la función de Rosenbrock para $n = 10$.*

La gráfica 1.18 muestra el error absoluto de los mejores individuos de cada ejecución y generación, la cual presenta la misma tendencia que los 2 ejemplos anteriores y una vez más demuestra que la convergencia del algoritmo no depende de la semilla aleatoria γ. Debido a que en este ejemplo tratamos con vectores de 10 entradas, nuestro AG requiere de más generaciones para poder ajustar cada una de ellas y aproximar el óptimo. Para las 30 ejecuciones el promedio del error absoluto fue de $\epsilon = 0.1915$, el cual es bastante aceptable considerando la rapidez con la que nuestro AG encontró el mejor individuo en 150,000 generaciones (alrededor de 20 segundos por ejecución).

El elevado número de generaciones necesarias ($G = 150,000$) para aproximar el óptimo global, en comparación a los demás ejemplos, no solo se debe al aumento en la dimensión del espacio de búsqueda ($n = 10$), también se debe al aumento en el parámetro η, el cual regula la lejanía con la que los hijos serán escogidos de los padres, favoreciendo hijos cercanos valores altos de η. Debido a que en este caso cada individuo está representado por un vector de tamaño 10, en lugar de solo un escalar como en los ejemplos anteriores, un cambio demasiado agresivo en cualquiera de las

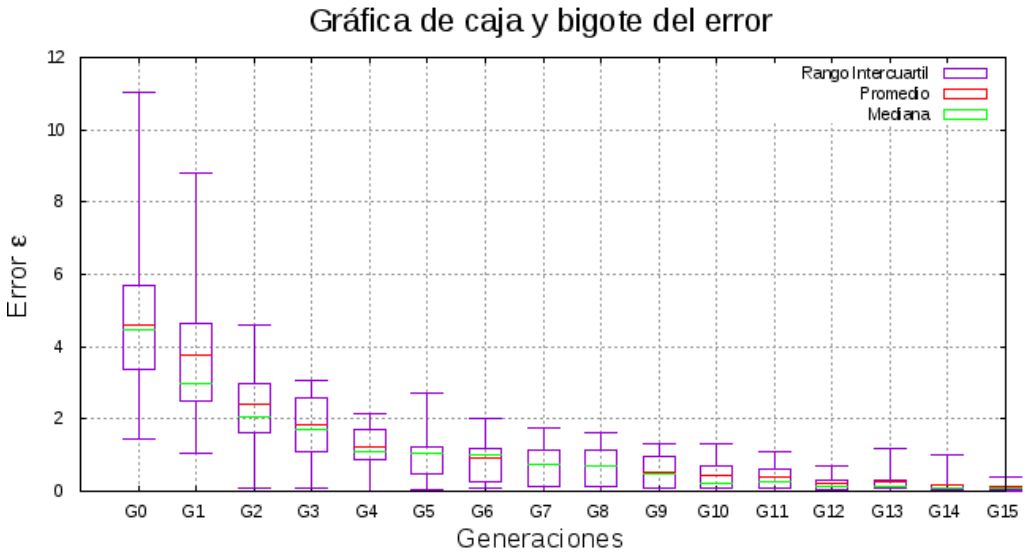
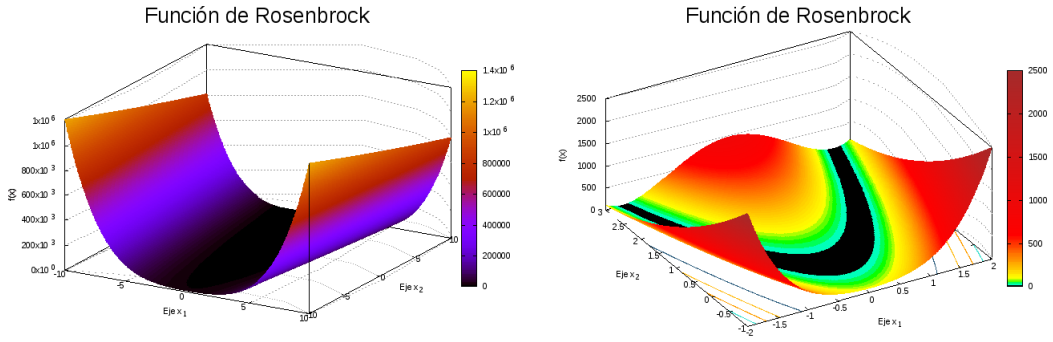


Figura 1.16: *Cada caja representa la distribución del error de los individuos mejor adaptados por generación en las 30 ejecuciones.*

entradas del vector podría significar la perdida de la línea de una buena solución. Por asegurarnos de que esto no suceda es necesario incrementar el parámetro η y así aumentar las probabilidades de que los descendientes sean escogidos cerca de los padres. En pocas palabras, si damos pasos más pequeños tendremos que caminar más para llegar a nuestro destino. En el siguiente capítulo se aborda el problema de optimización con múltiples objetivos y la adaptación del algoritmo genético para este caso.



(a) Función de Rosenbrock para $-10 \leq x_i \leq 10$ e $i = 1, 2$.

(b) Función de Rosenbrock para $-2 \leq x_1 \leq 2$ y $-1 \leq x_2 \leq 3$ (acercamiento)

Figura 1.17: *Gráficas de la función de Rosenbrock para* $m = 2$.

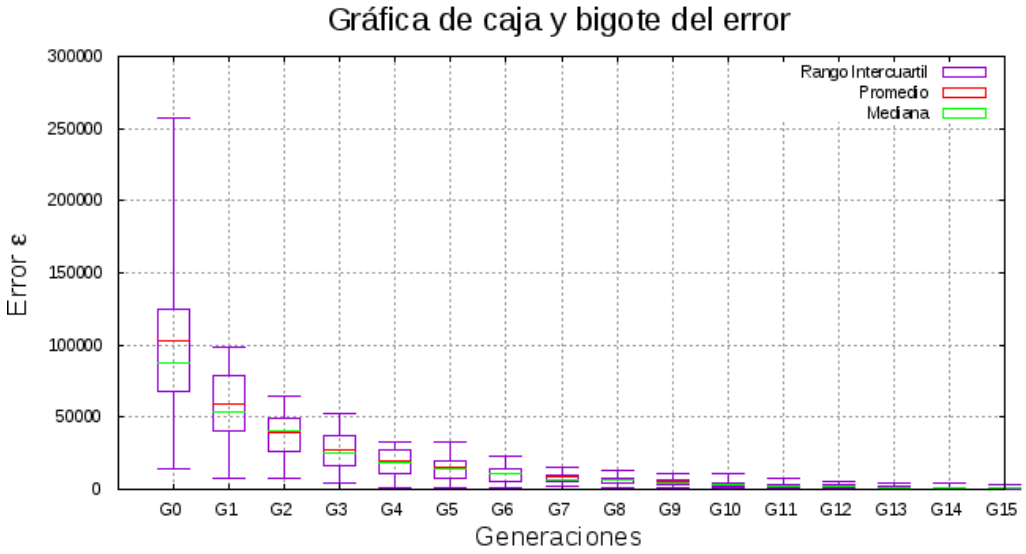


Figura 1.18: *Error absoluto de las 30 ejecuciones por generación para la función de Rosenbrock con* $n = 10$.

Capítulo 2

Algoritmo genético multiobjetivo: NSGA-II

2.1. Optimización multiobjetivo

Un problema de optimización multi-objetivo (Problema de Optimización Multi-Objetivo (MOP)) es aquel que tiene 2 o más funciones objetivo dispuestas a ser maximizadas o minimizadas. Al igual que en la optimización mono-objetivo, un MOP usualmente está sujeto a ciertas restricciones que debe cumplir. La definición 2.1 muestra la forma general de un MOP.

Definición 2.1. *Sea $f_k : \mathbb{R}^n \to \mathbb{R}$, para $k \in \{1, ..., K\}$. Un Problema de Optimización Multi-Objetivo MOP (por sus siglas en inglés) es aquel que tiene la siguiente forma:*

$$\begin{array}{rlll} \textit{Minimizar} & f_k(x), & & k = 1, 2, ..., K; \\ \textit{Sujeto a} & g_j(x) \geq 0, & & j = 1, 2, ..., J; \\ & h_q(x) = 0, & & q = 1, 2, ..., Q; \\ & x_i^L \leq x_i \leq x_i^S, & & i = 1, 2, ..., n; \end{array} \tag{2.1}$$

donde x_i^L, x_i^S son los limites inferior y superior de cada x_i respectivamente, para $i \in \{1, ..., n\}$.

A las funciones $f_k(x)$ las llamamos funciones objetivo, mientras que a las $J + Q$ funciones de la forma $g_j(x) \geq 0$ y $h_q(x) = 0$ las llamaremos funciones restrictivas, o simplemente *restricciones*. Las restricciones de la forma $x_i^L \leq x_i \leq x_i^S$ son llamadas límites para las variables o restricciones de caja. Como el lector ya habrá notado, la definición 2.1 solo contempla problemas de minimización, mientras que en el mundo real existen muchos problemas de maximización e incluso MOP que contemplan ambos. Esto no supone ninguna limitante al marco teórico que daremos en esta sección, ya que basta con multiplicar por -1 cada función que requiera ser maximizada para convertirla en una a ser minimizada. Lo mismo aplica para restricciones $g_j(x) \leq 0$.

La definición 2.1 es una generalización de la definición 1.1 vista en el capítulo 1. Note que a diferencia de ésta, la definición 2.1 muestra un problema cuyo objetivo es minimizar K funciones objetivo. Esto implica que, la solución a un MOP sera un vector $x \in \mathbb{R}^n$ en lugar de un punto en $x \in \mathbb{R}$.

Notación. *Denotaremos por $F(x)$ al vector de tamaño K formado por las funciones objetivo $f_k(x)$ de la definición 2.1 en cada una de sus entradas:*

$$F(x) := (f_1(x), f_2(x), ..., f_K(x))$$

Definición 2.2. *Sea $\mathcal{D} \subseteq \mathbb{R}^n$. Definimos la Región Factible $\mathcal{D}$ como el conjunto de vectores $x \in \mathbb{R}^n$ tales que cumplen las restricciones de la definición 2.1. A los vectores $x \in \mathcal{D}$ los llamaremos soluciones factibles.*

En lo subsecuente, cada vez que nos refiramos a una posible solución o individuo, estaremos hablando de un vector en la región factible $\mathcal{D}$. Cabe destacar que, en este trabajo abordaremos solo problemas de optimización sin funciones restrictivas y con $K = 2$ funciones objetivo.

Una de las diferencias fundamentales entre optimización mono y multi objetivo, es que en la optimización multi-objetivo las funciones objetivo constituyen un espacio multi-dimensional.

Notación. *Denotaremos por $F(\mathcal{D})$ al espacio de dimensión K formado por las funciones objetivo de la definición 2.1 valuadas en la región factible $\mathcal{D}$.*

$$F(\mathcal{D}) := (f_1(\mathcal{D}),\ f_2(\mathcal{D}), ...,\ f_K(\mathcal{D}))$$

Al espacio $F(\mathcal{D})$ lo llamaremos "espacio objetivo factible" o simplemente como "espacio de los objetivos".

2.1.1. Relación de dominancia

Como ya mencionamos anteriormente, en los problemas optimización mono-objetivo las posibles soluciones son valores escalares, por lo que siempre es posible comparar cualesquiera dos y decidir cuál es mejor que otra. Sin embargo, para un problema optimización multi-objetivo la comparación de posibles soluciones es llevada a cabo en $F(\mathcal{D}) \subseteq \mathbb{R}^K$ (con $K \geq 2$). Debido a que en general para $\mathbb{R}^n$ no se conoce una relación de orden total, para poder compara posibles soluciones de problemas de optimización multi-objetivo utilizaremos el concepto de *Dominancia de Pareto*, el cual definir una relación de orden parcial.

Definición 2.3 ([Mun99])**.** *Sea X un conjunto no vacío y $\mathcal{R}$ una relación binaria sobre X, decimos que:*

1. *$\mathcal{R}$ es Reflexiva si, $a\mathcal{R}a \qquad \forall\ a \in X$*
2. *$\mathcal{R}$ es Simétrica si, $a\mathcal{R}b \Rightarrow b\mathcal{R}a$*

3. $\mathcal{R}$ *es Antisimétrica si,* $a\mathcal{R}b$ *y* $b\mathcal{R}a \Leftrightarrow a = b$

4. $\mathcal{R}$ *es Asimétrica si,* $a\mathcal{R}b \Rightarrow \neg(b\mathcal{R}a)$ *(b no está relacionado con a)*

5. $\mathcal{R}$ *es Transitiva si,* $a\mathcal{R}b$ *y* $b\mathcal{R}c \Rightarrow a\mathcal{R}c$

Definición 2.4 ([Mun99]). *Sea X un conjunto no vacío y $\mathcal{R}$ una relación binaria sobre X. Decimos $\mathcal{R}$ es un orden parcial en X si:*

- $\mathcal{R}$ *es Reflexiva.*
- $\mathcal{R}$ *es Antisimétrica.*
- $\mathcal{R}$ *es Transitiva.*

Entonces el par $(X, \mathcal{R})$ se llama conjunto parcialmente ordenado.

Definición 2.5 ([Mun99]). *Sea $(X, \mathcal{R})$ un conjunto parcialmente ordenado, si se cumple la siguiente condición:*

- $a\mathcal{R}b$ *o* $b\mathcal{R}a \qquad \forall\, a, b \in X$ *(Completitud)*

Entonces el par $(X, \mathcal{R})$ se llama conjunto totalmente ordenado.

Observación. *Note que es posible aplicar la definición 2.3 en conjuntos $X \subset \mathbb{R}^n$ cuyos elementos sean vectores de dimensión $n \geq 2$. Es decir, basta con definir la relación binaria $\mathcal{R}$ para cualesquiera 2 vectores $x = (x_1, x_2, ..., x_n),\ y = (y_1, y_2, ..., y_n) \in X$ como sigue:*

$$x\mathcal{R}y \quad \Leftrightarrow \quad x_i\mathcal{R}y_i \qquad \forall\, i = 1, 2, ..., n$$

Para problemas de optimización con un solo objetivo, $f(\mathcal{D})$ usualmente está contenido en $\mathbb{R}$, el cual es un conjunto totalmente ordenado; es decir, siempre se puede decidir cuando un elemento es menor que otro. Sin embargo, para un MOP $F(\mathcal{D})$ está usualmente contenido en $\mathbb{R}^n$, con $n \geq 2$ (o cualquier sub-espacio de $\mathbb{R}$ de dimensión n), el cual es un conjunto parcialmente ordenado sobre la relación binaria $\leq$ componente a componente (observación anterior). Ya que existen elementos $x, y \in \mathbb{R}^n$ tales que no es posible determinar si $x \leq y$ o $y \leq x$, por ejemplo $x = (1, 2)$ y $y = (2, 1)$. Para ilustrar ésto, a continuación damos un ejemplo muy simple de un MOP.

Suponga que se desea comprar una computadora personal. Si usted fuera una persona para la cual el dinero no es problema, entonces compraría la computadora con el mejor rendimiento disponible en el mercado. O si usted la necesitara y pudiera gastar solo lo mínimo en ella entonces compraría la de menor costo. Pero si usted es una persona promedio, entonces podría darse el lujo de elegir alguna entre estos dos extremos. En este caso podríamos ver a f_1 como el costo del equipo, mientras que a f_2 como el rendimiento. Notará que a nosotros nos interesa comprar la computadora con el mejor rendimiento posible (maximizar f_2) sin sacrificar una fortuna (minimizar

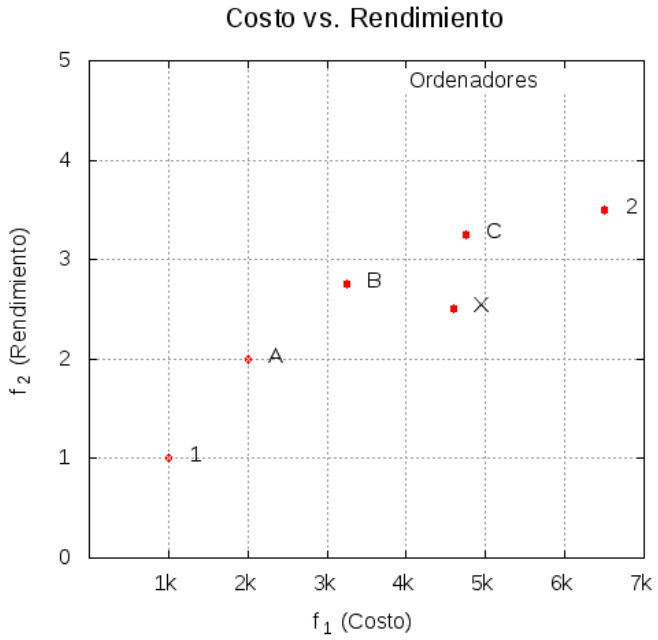


Figura 2.1: *Gráfica del costo VS rendimiento para los diferentes equipos.*

f_1), es decir, ambas funciones objetivo f_1 y f_2 se contraponen. Para que esto quede un poco más claro observe la figura 2.1 y la tabla 2.1.

Como puede observarse en la tabla 2.1 y la figura 2.1, los equipos 1 y 2 representan los extremos en el mercado. Es decir, el equipo 1 es el más barato, pero también es el que ofrece menor rendimiento, mientras que el equipo 2 es el de mejor rendimiento, pero también el más caro. Note también que la escala de precio va de \$1,000 a \$4,600 y la de rendimiento del 1 al 3.5, siendo 1 el peor rendimiento y 3.5 el mejor.

Entonces, si nosotros quisiéramos adquirir un equipo que no fuese el de mejor rendimiento por ser el más caro, pero tampoco el más barato por ser el de peor rendimiento, ¿cuál compraríamos? Esto nos deja 4 opciones, los equipos A, B, C o X. En cuanto a los ordenadores A, B, C no es posible decidir bajo nuestros criterios cuál de ellos es el mejor, debido a que siempre que los comparemos alguno de los dos ganará en costo, pero perderá en rendimiento (ya que la relación binaria $\leq$ no cumple la condición de completitud sobre $F(\mathcal{D})$). Es decir, nuestras funciones objetivo f_1 y f_2 se contraponen. Sin embargo, existe un equipo que si podemos descartar respecto a los otros, el equipo X. Note que al comparar los equipos X y C nos encontramos con el mismo dilema, el equipo X es menos costoso que el equipo C, pero el rendimiento de C es mejor que el de X, y lo contrario sucede al comparar a los equipos X y A. Pero al comparar al equipo X con el B nos encontramos con que el equipo B supera al X en ambos objetivos, es decir, el equipo B es una mejor opción que X. En este caso diremos que la solución B "domina" a la solución X.

El análisis anterior nos lleva a la conclusión de que dado un MOP, es posible

Equipo	f_1 (Costo)	f_2 (Rendimiento)
1	1.00k	1
A	2.00k	2
B	3.25k	2.75
C	4.75k	3.25
2	6.50k	3.5
X	4.60k	2.5

Tabla 2.1: *Costo y rendimiento de los equipos.*

dividir la región factible $\mathcal{D}$ en dos conjuntos fundamentales, siendo el primero aquel que contenga todas las soluciones que no es posible comparar entre si {1, A, B, C, 2}, y el segundo conjunto aquel que contenga a las soluciones que, al ser comparadas con los elementos del otro conjunto, fueron peores contra al menos uno elemento del primer conjunto, es decir, {X}. Para ello, debemos primero definir una relación binaria sobre $F(\mathcal{D})$ con el fin de otorgar cierto orden y jerarquía a las posibles soluciones.

Definición 2.6. *Sea $\mathcal{R}$ una relación binaria sobre un conjunto no vacío X. Diremos que $\mathcal{R}$ es un orden parcial estricto sobre X si:*

- *$\mathcal{R}$ no es Reflexiva.*
- *$\mathcal{R}$ es Asimétrica.*
- *$\mathcal{R}$ es Transitiva.*

Observación. *Note que la relación $<$ define un orden parcial estricto sobre $F(\mathcal{D})$.*

Definición 2.7. *Sean $x, y \in \mathcal{D}$.*

- *Diremos que "x" es una mejor solución que "y" con respecto a la función objetivo f_k, si $f_k(x) < f_k(y)$ siempre que f_k deba ser minimizada, con $k = 1, 2, ..., K$.*
- *Diremos que "y" es una mejor solución que "x" con respecto a la función objetivo f_k, si $f_k(y) > f_k(x)$ siempre que f_k deba ser maximizada, con $k = 1, 2, ..., K$.*

Notación. *Ya sea cualquiera de los dos casos de la definición 2.7, utilizaremos el símbolo $\lhd$ para denotar que la solución "x" es mejor que "y" respecto a la función objetivo f_k, es decir $f_k(x) \lhd f_k(y)$*

Definición 2.8. *Sean $x, y \in \mathcal{D}$. Diremos que la solución "x" domina a la solución "y" en el sentido de Pareto, si las siguientes dos condiciones se cumplen:*

1. *La solución "x" no es peor que "y" en todas las funciones objetivo. Es decir,* $f_k(x) \not\triangleright f_k(x) \ \ \forall\ k = 1, 2, ..., K$.

2. *La solución "x" es estrictamente mejor que y en al menos una función objetivo. Es decir,* $\exists\ k \in \{1, 2, ..., K\}$ *tal que* $f_k(x) \triangleleft f_k(x)$.

Notación. *Si la solución "x" cumple con las dos condiciones de la definición 2.8 sobre la solución "y", entonces diremos que "x" domina a "y" y lo denotaremos como* $x \preceq y$.

Definición 2.9. *Sean* $x, y \in \mathcal{D}$. *Diremos que la solución "x" domina* ***fuertemente*** *a la solución "y", si la solución "x" es estrictamente mejor que la solución "y" en todas las funciones objetivo. Es decir,* $f_k(x) < f_k(y)\ \forall\ k = 1, 2, ..., K$.

Notación. *Si la solución "x" domina fuertemente a la solución "y", entonces lo denotaremos como* $x \prec y$.

Note que de la definición 2.8 (respectivamente definición 2.9) podemos afirmar que dadas cualesquiera 2 soluciones $x, y \in \mathcal{D}$, solo una de las siguientes posibilidades puede ocurrir:

- x domina a y, $x \preceq y$ (respectivamente $x \prec y$).
- y domina a x, $y \preceq x$ (respectivamente $y \prec x$).
- x no domina a y y y no domina a x (respectivamente $x \not\prec y$ y $y \not\prec x$).

Observación. *Toda relación binaria* $\mathcal{R}$ *que sea al menos Transitiva sobre un conjunto X, califica como una relación de orden.*

De la observación anterior note que la relación binaria $\prec$ es transitiva sobre el conjunto $\mathcal{D}$, por lo que esta califica como una relación de orden sobre $\mathcal{D}$. Más aún, dados cualesquiera $x, y \in \mathcal{D}$, $x \not\prec x$, es decir, $\not\prec$ es no reflexiva, además si $x \prec y$ entonces, $y \not\prec x$, por lo que $\prec$ también es asimétrica. Por lo tanto, $\prec$ es un orden parcial estricto sobre $\mathcal{D}$.

Ya que tenemos bien establecido el concepto de "Dominancia" entre cualesquiera dos puntos de $\mathcal{D}$, podemos dividir formalmente a este en 2 conjuntos fundamentales [DK01].

Definición 2.10. *Al conjunto cuyos elementos no sean dominados fuertemente por ningún otro elemento de* $\mathcal{D}$ *lo llamaremos "conjunto de elementos no-dominados" y lo denotaremos como:*

$$P_1 := \{x \in \mathcal{D} \mid \nexists\ y \in \mathcal{D}\ \ni\ y \prec x\}$$

Al conjunto cuyos elementos sean dominados fuertemente por al menos un elemento de P_1 lo llamaremos "conjunto de elementos dominados" y lo denotaremos como:

$$P_2 := \{y \in \mathcal{D} \mid \exists\ x \in P_1\ \ni\ x \prec y\}$$

De la definición anterior no está de más resaltar que dado cualquier elemento $y \in P_2$ siempre existirá al menos un elemento $x \in P_1$, tal que $x \prec y$. Es decir, P_1 tiene la propiedad de dominar a toda solución de P_2.

Observación. $\mathcal{D} = P_1 \cup P_2$, $\varnothing = P_1 \cap P_2$

En el ejemplo de las computadoras no era posible determinar cuál de las posibles soluciones en el conjunto {1, A, B, C, 2} era mejor con respecto a las otras, y lo mismo ocurrió al comparar el equipo X con el A y el C. Sin embargo, al comparar B con X nos encontramos que B$\prec$X. Es decir, en este ejemplo el conjunto de elementos no-dominados fue P_1 ={1, A, B, C, 2}, mientras que el conjunto de elementos dominados fue P_2 ={X} y el hecho de que el equipo A y C no dominaran a X no significo que X perteneciera a P_1, por lo que para que un elemento pertenezca a P_1 este no debe ser dominado por ningún elemento de P_1.

El hecho de que para los MOP's el contradominio $F(\mathcal{D})$ no sea un conjunto totalmente ordenado conlleva ciertas complicaciones en los operadores de nuestro AG tradicional, específicamente en el operador de selección. Ya que tendremos casos en los que no será posible decir cuando una solución es mejor que otra, y no podremos favorecerla o descartarla para su selección. La solución a este problema, entre otros, será explicada en las secciones subsecuentes, llegado el momento de abordar el tema central de este trabajo, el algoritmo Non-dominated Sorting Genetic Algorithm II (NSGA-II).

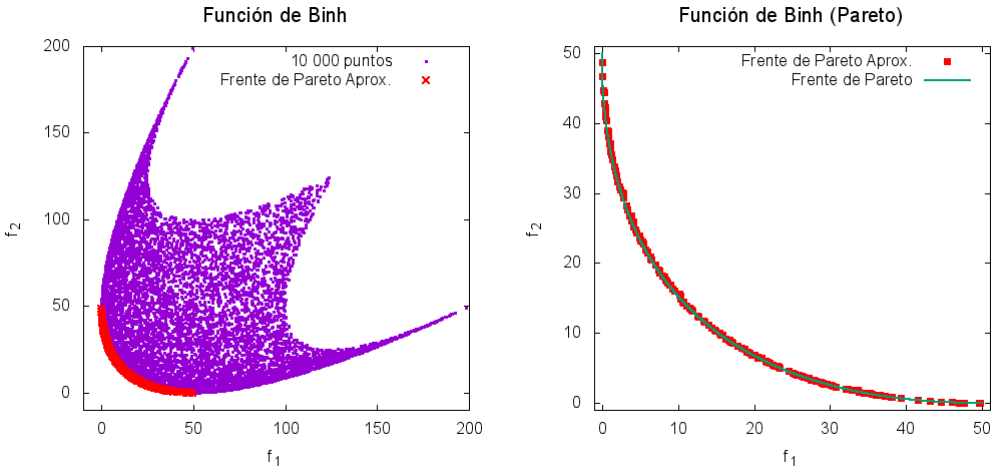
Ya que tenemos bien definido el concepto de "dominancia" y separado a $\mathcal{D}$ con respecto a la relación binaria $\prec$, es momento de definir el "frente de Pareto".

2.1.2. Optimalidad de Pareto

Considere el MOP de prueba denotado por BINH1 [DgB95], cuya gráfica es mostrada en la figura 2.2:

$$\begin{aligned} \textit{Minimizar} \quad & F = (f_1(x,y), f_2(x,y)),\ \textit{donde} \\ & f_1(x,y) = x^2 + y^2, \\ & f_2(x,y) = (x-5)^2 + (y-5)^2 \\ \textit{Sujeto a} \quad & -5 \leq x, y \leq 10 \end{aligned} \tag{2.2}$$

Para hacer las gráficas de la figura 2.2 se generaron 10,000 puntos aleatorios en el dominio del problema (cuadros moradas y equis rojas). Posteriormente por medio de un algoritmo (cuyo funcionamiento no es de mayor relevancia en este trabajo) fueron



(a) BINH1 (10,000 ptos. aleatorios). **(b)** Puntos y frente de Pareto de BINH1.

Figura 2.2: *Gráficas del MOP BINH1.*

encontrados los puntos no-dominados, cuyos elementos en el contradominio constituyen las soluciones optimas en el MOP (cuadros rojos). Para poder explicar mejor el concepto de optimalidad en un MOP, daremos a continuación algunas definiciones.

Definición 2.11 (Conjunto Global de Pareto)**.** *Dado un MOP definimos el Conjunto Global de Pareto $\mathcal{P}^*$, como el conjunto de elementos no-dominados de toda la región factible $\mathcal{D}$, es decir P_1, por lo que:*

$$\mathcal{P}^* := \{x \in \mathcal{D} \mid \nexists\, y \in \mathcal{D} \text{ tal que } y \prec x\}$$

En lo subsecuente nosotros nos referiremos al Conjunto Global de Pareto simplemente como conjunto de Pareto.

Definición 2.12 (Frente de Pareto)**.** *Dado un MOP con función objetivo $F(x)$ y conjunto de Pareto $\mathcal{P}^*$, definimos el Frente de Pareto como:*

$$\mathcal{FP}^* := \{u = F(x) \mid x \in \mathcal{P}^*\}$$

Como seguramente el lector ya habrá vislumbrado, el objetivo de un Algoritmo Evolutivo Multi-Objetivo (MOEA) es aproximar el frente de Pareto para un MOP dado. Sin embargo, los conjuntos $\mathcal{P}^*$ y $\mathcal{FP}^*$ son no finitos (por lo general no numerables, y en el mejor de los casos numerables), mientras que computacionalmente hablando solo podemos tratar con conjuntos finitos, con además, elementos de longitud finita. Por ello es necesario establecer una notación que nos permita diferenciar entre el conjunto de Pareto y frente de Pareto teóricos y sus respectivos conjuntos prácticos o valores computacionales.

Notación. *Denotaremos por* $\mathcal{P}_{aprox}$ *y* $\mathcal{FP}_{aprox}$*, al conjunto de Pareto y el frente de Pareto aproximados, es decir, a aquellos que fueron calculados por medio de una computadora.*

Retomando el problema denotado por BINH1, en la figura 2.2 los cuadros rojos representan el frente de Pareto aproximado, es decir $\mathcal{FP}_{aprox}$, que son el contradominio del conjunto de Pareto aproximado $\mathcal{P}_{aprox}$ (figura 2.3). Como seguramente el lector habrá notado, en la figura 2.2 b también aparece una curva o línea de color verde que pasa por encima de los cuadros rojos, dicha curva representa el frente de Pareto real, es decir $\mathcal{FP}^*$.

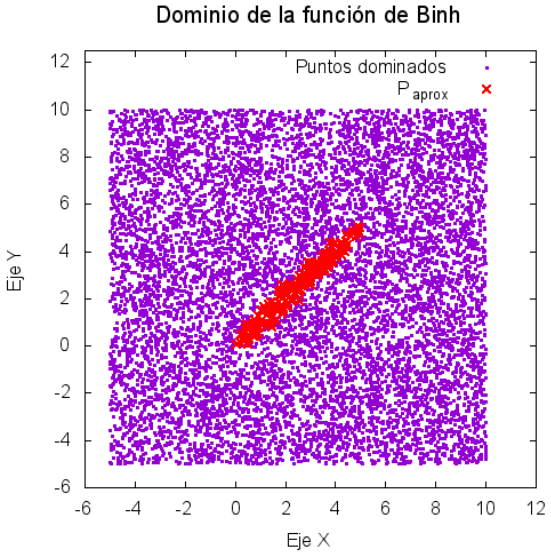


Figura 2.3: *Dominio del MOP BINH1 y puntos de Pareto aproximados.*

Como puede observarse en la figura 2.3, los puntos (o equis) de Pareto aproximados están cercanos o sobre la recta $y = x$ para $x \in [0, 5]$, es decir el frente de Pareto real $\mathcal{FP}^*$ (curva verde) se da cuando $y = x$ y $x \in [0, 5]$.

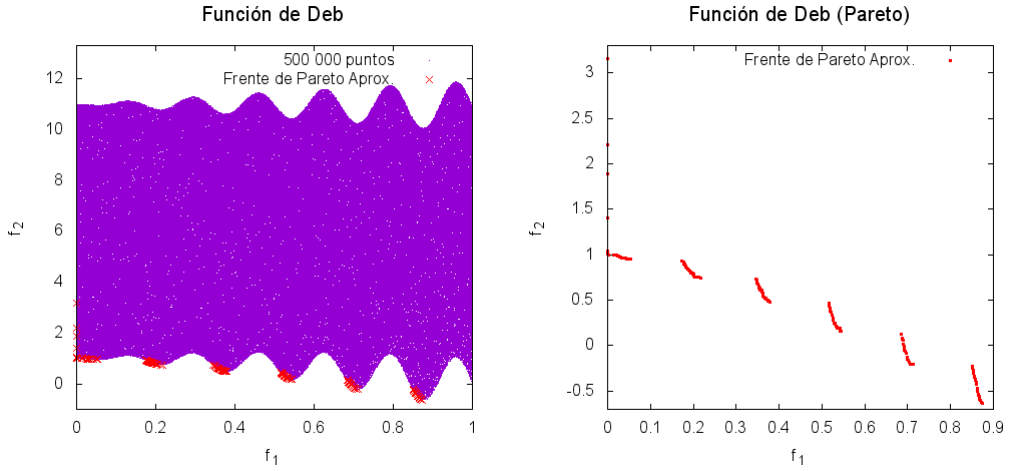
Como puede apreciarse en las figuras 2.2 b, las aproximaciones de $\mathcal{FP}_{aprox}$ a $\mathcal{FP}^*$, es buena considerando 3 aspectos fundamentales: proximidad, distribución y extensión. Es decir, los puntos de $\mathcal{FP}_{aprox}$ se aproximan bastante bien a $\mathcal{FP}^*$, se distribuyen uniformemente a lo largo de $\mathcal{FP}^*$ y se extienden por todo lo largo de $\mathcal{FP}^*$.

Como puede notar en la figura 2.2 b, el $\mathcal{FP}^*$ del MOP BINH1 es una curva continua, sin embargo, en muchos MOP's esto no ocurre así.

Considere el siguiente MOP de prueba denotado como DEB2[1]:

$$
\begin{aligned}
&Minimizar \\
&\quad F = (f_1(x), f_2(x,y)),\ donde \\
&\quad f_1(x) = x, \\
&\quad f_2(x,y) = (1+10y)\left[1 - \left(\frac{x}{1+10y}\right)^2 - \frac{x}{1+10y} sen(12\pi x)\right] \\
&Sujeto\ a \quad 0 \leq x, y \leq 1
\end{aligned}
\tag{2.3}
$$

cuya gráfica es mostrada en la figura 2.4.



(a) DEB2 (10,000 ptos. aleatorios).

(b) Puntos y frente de Pareto del MOP DEB2.

Figura 2.4: *Gráficas del MOP DEB2.*

Como puede observarse en las figuras 2.4 a y b, el MOP denotado como DEB2 es bastante más complicado que el MOP BINH1, al menos en cuanto al $\mathcal{FP}_{aprox}$ se refiere, ya que como puede apreciarse este no se encuentra sobre una curva continua y a medida que el valor de x avanza los elementos de $\mathcal{FP}_{aprox}$ se van separando más, a pesar de que en esta ocasión se graficaron 500,000 puntos aleatorios.

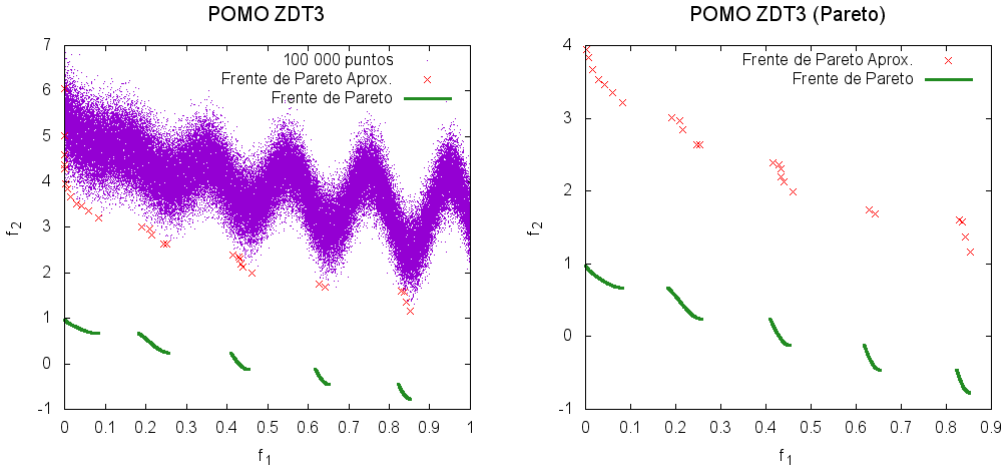
Esto sugiere que existen funciones que, al aplicarlas al dominio, los puntos en el contradominio no estarán distribuidos de forma tan uniforme como en el dominio. En el MOP DEB2 esto no es muy notorio, sin embargo, la función objetivo del siguiente MOP de prueba ejemplifica esto mucho mejor.

[1]Este MOP puede ser consultado en la página http://www.cs.cinvestav.mx/~emoobook/, apéndice A, problema: DEB (2), o con más detalle en el libro [DgB95] como: MOP6.

Considere el siguiente MOP de prueba denotado como ZDT3 [DgB95]:

$$
\begin{aligned}
&\textit{Minimizar} \\
&\quad F = (f_1(x), f_2(x)),\ \textit{donde} \\
&\quad f_1(x) = x_1, \\
&\quad f_2(f_1, g) = g(x)\left[1 - \sqrt{\frac{f_1(x)}{g(x)}} - \frac{f_1(x)}{g(x)} sen(10\pi f_1(x))\right] \qquad (2.4) \\
&\quad y \\
&\quad g(x) = 1 + \frac{9}{n-1}\sum_{i=2}^{n} x_i \\
&\textit{Sujeto a} \qquad 0 \leq x_i \leq 1, \quad i = 1, 2, ..., n
\end{aligned}
$$

donde $x = (x_1, x_2, ..., x_n)$, $n = 30$ y cuya gráfica es mostrada en la figura A.1.



(a) ZDT3 (100,000 ptos. aleatorios).

(b) Puntos y frente de Pareto del MOP ZDT3.

Figura 2.5: *Gráficas del MOP ZDT3.*

La presencia de la función $sen()$ en f_2 causa discontinuidades en $\mathcal{FP}^*$, a pesar de que estas no existen en la región factible $\mathcal{D}$.

En el MOP ZDT3, el $\mathcal{FP}^*$ es formado cuando $g(x) = 1$ (es decir, cuando $x_i = 0\ \forall\ i = 2, 3, ..., n$) y $x_1 \in A$.
donde

$$
\begin{aligned}
A = &[0, 0.0830015349] \cup \\
&(0.1822287280, 0.2577623634] \cup \\
&(0.4093136748, 0.4538821041] \cup \\
&(0.6183967944, 0.6525117038] \cup \\
&(0.8233317983, 0.8518328654]
\end{aligned}
$$

Sin embargo, el frente de Pareto Real del ZDT3, no es ciertamente el especificado arriba, es decir $\mathcal{FP}^*$, ya que los límites de los intervalos del conjunto A son aproximados. Aun así, denotaremos esta aproximación del frente de Pareto real del ZDT3 como $\mathcal{FP}^*$ a falta de uno mejor.

Como puede observarse en las figuras A.1 a y b, a pesar de que se generaron 100,000 puntos aleatorios uniformente distribuidos en el dominio, al aplicar la función objetivo, ésta concentro los puntos alrededor de un senoide en el contradominio (franja formada por los cuadros morada). Esto provoca que nuestra aproximación del frente de Pareto $\mathcal{FP}_{aprox}$ quede muy lejana a $\mathcal{FP}^*$, sin mencionar la baja cantidad de soluciones en $\mathcal{P}_{aprox}$.

Funciones como la ZDT3 demuestran que existen problemas que requieren más que una simple búsqueda aleatoria en el contradominio de la función objetivo para encontrar el $\mathcal{FP}^*$.

2.2. Algoritmo evolutivo multiobjetivo

La primera aplicación real de un AE a un MOP para encontrar múltiples soluciones fue propuesta y desarrollada por David Schaffer en su tesis doctoral (1984) y consistía básicamente en una modificación simple de un AG mono-objetivo, a la cual llamo Vector Evaluated Genetic Algorithm (VEGA) [ES03]. A pesar de que si éste se ejecutaba durante un gran número de iteraciones la población convergía a un óptimo individual, el VEGA demostró de manera práctica que era posible resolver MOP's por medio de un AG. Para 1989, David E. Goldberg sugirió en su libro [Gol89] que era posible construir un MOEA[2] en tan solo 10 líneas basándose en el concepto de "Dominancia" (el cual explicaremos ampliamente en los siguientes párrafos). A partir del trabajo de Goldberg, numerosos investigadores han diseñado y desarrollado diferentes tipos de MOEA's, tales como el Strength Pareto Evolutionary Algorithm II (SPEA-II) [ZLT01] o el Multiobjective Selection Based on Dominated Hypervolume (SMS-EMOA) [BNE07]. En particular, este trabajo se enfoca en el MOEA conocido como NSGA-II[3].

En 1994 N. Srinivas y Kalyanmoy Deb [SD94] desarrollaron una AG basado la clasificación de elementos no dominados (Nondominated Sorting) sugerida por Goldberg en 1989 [Gol89]. Debido a este antecedente lo llamaron "Nondominated Sorting Genetic Algorithm" (Non-dominated Sorting Genetic Algorithm (NSGA)) [SD94]. Sin embargo, el NSGA presentaba ciertas deficiencias, tales como la falta de elitismo y una complejidad computacional alta[4]. Dados estos problemas, en el 2001 Kalyanmoy Deb presento la segunda versión del NSGA, que a pesar de ser muy diferente por las mejoras que este presentaba, decidió llamarlo NSGA-II el cual, se

[2]Acrónimo en inglés de Multi-Objective Evolutionary Algorithms.

[3]El algoritmo NSGA-II fue desarrollado por el Prof. Kalyanmoy Deb en conjunto con otros investigadores en el Laboratorio de Algoritmos Genéticos de Kanpur, India y puede ser consultado con mayor detalle en [DPAM02].

[4]Se le llama complejidad computacional al número de cálculos realizados por un algoritmo

convertiría en el AG más citado y comparado por los investigadores en el ramo de la Programación Evolutiva (PE) [DPAM02], así como el objeto central de este trabajo.

Como su nombre lo indica, el NSGA-II se basa en la clasificación de elementos no dominados, para encarar la dificultad de decidir cuándo una solución (o grupo de soluciones) es mejor que otra (otras), presente en los MOPs. Algunas veces la clasificación de elementos no dominados no es suficiente para decir cuando una solución es mejor que otra, ya que básicamente se basa en dividir a la población en subconjuntos disjuntos, por lo que cuando los individuos están en el mismo subconjunto es necesario aplicar otro criterio. Este segundo criterio ordenara a los elementos dependiendo en la densidad de la población alrededor de estos, y escoge a aquellos que estén en zonas menos densamente pobladas. Al valor de densidad sobre el que se basa éste último criterio se le llama "Crowding Distance".

El segundo criterio antes mencionado también funciona como un mecanismo para preservar la diversidad en las soluciones seleccionadas, evitando que estas se concentren en alguna región. Antes de abordar el algoritmo NSGA-II por completo, primero definiremos los dos operadores mencionados anteriormente y los ilustraremos con un pequeño ejemplo numérico.

2.2.1. Clasificación veloz de elementos no dominados

La clasificación veloz de elementos no dominados (Fast Nondominated Sorting Approach) tiene como objetivo dividir a una población P con λ elementos, en subconjuntos disjuntos llamados frentes $\mathcal{F}_i$ con $1 \leq i \leq \lambda$, donde cada frente contiene elementos de P con un mismo nivel de dominancia. Es decir, dada la población P ésta es analizada para encontrar los elementos no dominados. Una vez identificados, éstos pasan a formar el primer frente $\mathcal{F}_1$, posteriormente los elementos de $\mathcal{F}_1$ son excluidos de P para proceder a encontrar el segundo frente $\mathcal{F}_2$. Es decir, el frente $\mathcal{F}_2$ contiene a los elementos no dominados de $P \backslash \mathcal{F}_1$. El proceso es repetido para encontrar $\mathcal{F}_3$ en el conjunto $P \backslash (\mathcal{F}_1 \cup \mathcal{F}_2)$ y así sucesivamente, hasta haber calcificado cada elemento de P en su respectivo frente $\mathcal{F}_i$ con $1 \leq i \leq \lambda$. Los dos casos extremos son cuando los λ elementos de P son no dominados, formando un solo frente $\mathcal{F}_1$ y cuan se tienen λ frentes, debido a que cada uno de ellos contiene un solo elemento de P.

Existen distintos algoritmos para realizar esta clasificación, algunos más veloces que otros. El algoritmo usado por el NSGA-II es llamado "Fast Nondominated Sorting Approach" debido a que éste es más eficiente y veloz comparado con el del NSGA original. El funcionamiento es mostrado en el algoritmo 4 y explicados en el siguiente párrafo.

Primero, para cada individuo $p \in P$ es necesario calcular su contador n_p, que es el número de soluciones que dominan a p, y el conjunto de soluciones a las que p domina, es decir S_p. Las líneas 1 a 15 del algoritmo 4 se encargan del procedimiento anterior, además de encontrar el primer frente $\mathcal{F}_1$, donde sus elementos son aquellos cuyo contador n_p es igual a 0. Notara que existe también una variable llamada p_{rank}, ésta especifica el "rango" o frente donde se encuentra, siendo 1 el más importante,

Algoritmo 4: clasificación_veloz_de_elementos_no_dominados(P) [DPAM02].

Input: Población P con λ elementos.
Output: Frentes $\mathcal{F}_i$ con $1 \leq i \leq \lambda$.

```
for each p ∈ P do
    S_p = ∅
    n_p = 0
    for each q ∈ P do
        if p ≺ q then
            S_p = S_p ∪ {q}
        else if q ≺ p then
            n_p = n_p + 1
        end
    end
    if n_p = 0 then
        p_rank = 1
        F_1 = F_1 ∪ {p}
    end
end
i = 1
while F_i ≠ ∅ do
    Q ≠ ∅
    for each p ∈ F_i do
        for each q ∈ S_p do
            n_q = n_q − 1
            if n_q = 0 then
                q_rank = i + 1
                Q = Q ∪ {q}
            end
        end
    end
    i = i + 1
    F_i = Q
end
return (F_1, F_2,...,F_i)
```

2 el siguiente más importante y así sucesivamente. Luego entre las líneas 16 a 30, primero visitamos cada elemento $q \in S_p$ de cada solución $p \in \mathcal{F}_1$, de tal modo que cada vez que se visita una solución q su contador n_q se reduce en 1. Las soluciones q cuyo contador n_q es igual a cero después de dicho procedimiento pasan a formar parte del segundo frente $\mathcal{F}_2$. El procediendo anterior es realizado nuevamente con los elementos $q \in S_p$ de las soluciones $p \in \mathcal{F}_2$ para encontrar $\mathcal{F}_3$. El procedimiento anterior se lleva a cabo hasta que todos los frentes sean encontrados, es decir, hasta que el rango de cada solución haya sido calculado.

Para clarificar el funcionamiento de este método de clasificación a continuación damos un pequeño ejemplo.

Ejemplo 2.1.

$$\begin{aligned} &\textit{Minimizar} \\ &\qquad F = (f_1(x), f_2(x,y)),\ \textit{donde} \\ &\qquad f_1(x) = x, \\ &\qquad f_2(x,y) = \frac{1+y}{x} \\ &\textit{Sujeto a} \quad 0.1 \leq x \leq 1 \\ &\qquad\qquad 0 \leq y \leq 5 \end{aligned} \tag{2.5}$$

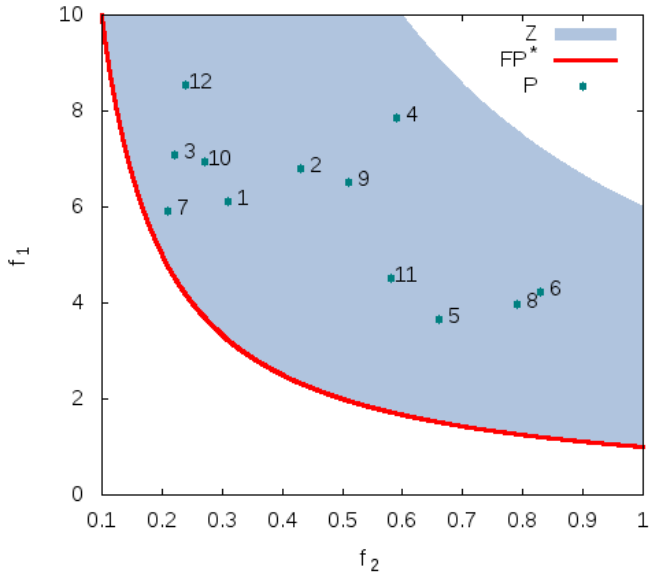


Figura 2.6: *Subconjunto del espacio objetivo factible junto con el frente de Pareto y una población aleatoria.*

La figura 2.6 muestra el subconjunto de $F(\mathcal{D})$ donde se encuentra el frente de Pareto $\mathcal{FP}^*$ del ejemplo 2.1. Además, esta figura muestra una población P con $\lambda = 12$ elementos aleatorios, los cuales pueden ser consultados en la tabla 2.2.

Población P						
Individuo	f_1	f_2		Individuo	f_1	f_2
$p_1 = (0.31, 0.89)$	0.31	6.10		$p_7 = (0.21, 0.24)$	0.21	5.90
$p_2 = (0.43, 1.92)$	0.43	6.79		$p_8 = (0.79, 2.14)$	0.79	3.97
$p_3 = (0.22, 0.56)$	0.22	7.09		$p_9 = (0.51, 2.32)$	0.51	6.51
$p_4 = (0.59, 3.63)$	0.59	7.85		$p_{10} = (0.27, 0.87)$	0.27	6.93
$p_5 = (0.66, 1.41)$	0.66	3.65		$p_{11} = (0.58, 1.62)$	0.58	4.52
$p_6 = (0.83, 2.51)$	0.83	4.23		$p_{12} = (0.24, 1.05)$	0.24	8.54

Tabla 2.2: *Población aleatoria P del ejemplo 2.1.*

Comencemos con la primera parte del algoritmo, correspondiente a las líneas 1 a 15 del pseudocódigo 4. El objetivo de esta primera parte es encontrar el contador n_p y el conjunto S_p para cada $p \in P$, y al mismo tiempo hallar el primer frente $\mathcal{F}_1$ conformado por los $p \in P$ tales que $n_p = 0$.

1. Tome el primer elemento de la tabla 2.2, es decir p_1 y haga $S_{p_1} = \varnothing$, $n_{p_1} = 0$ (líneas 2 y 3 del pseudocódigo). Luego, tome $q = p_2$ (línea 4 del pseudocódigo) y compárelo con p_1 bajo la relación $\prec$. Como $f_1(p_1) = 0.31 < f_1(p_2) = 0.43$ y $f_2(p_1) = 6.10 < f_2(p_2) = 6.79$, entonces $p_1 \prec p_2$ (línea 5 del pseudocódigo). Por lo tanto $S_{p_1} = \varnothing \cup \{p_2\}$ (línea 6 del pseudocódigo).

2. Como indica la línea 4 del pseudocódigo, realice la comparativa de p_1 con los demás elementos $p_j \in P$, con $j = 3, 4, ..., 12$. Al terminar notara que $S_{p_1} = \{p_2, p_4, p_9\}$ y $n_{p_1} = 1$.

3. Por último realice los dos puntos anteriores con los demás elementos $p \in P$ como indica la línea 1 del pseudocódigo, de tal forma que al término obtenga los conjuntos S_p y los coeficientes n_p de cada elemento de la población, como puede apreciarse en la tabla 2.3. Adicional a esto, establezca el parámetro $p_{rank} = 1$ para los $p \in P$ tales que $n_p = 0$.

Como puede observarse en la tabla 2.3, el contador n_p de los individuos $p_5,\ p_7,\ p_{11}$ es 0, por lo que estos conformaran los elementos del primer frente, es decir $\mathcal{F}_1 = \{p_5, p_7, p_{11}\}$. Note que la afirmación anterior concuerda que la figura 2.6. Además, observe que el ranking o clasificación de los elementos en $\mathcal{F}_1$ es igual al índice del frente, es decir $p_{rank} = 1 \ \forall \ p \in \mathcal{F}_1$.

La segunda parte del algoritmo corresponde a las líneas 16 a 30 del pseudocódigo 4. En estas, a partir del último frente conocido y de los parámetros S_p y n_p de cada uno de sus individuas, se determinará el siguiente frente. Es decir, utilizando a los individuos del ya conocido $\mathcal{F}_1$ y sus respectivos parámetros determinaremos $\mathcal{F}_2$. Luego con $\mathcal{F}_2$ y los parámetros de sus integrantes determinaremos $\mathcal{F}_3$, y así sucesivamente

Población P						
Individuo	S_p	n_p		Individuo	S_p	n_p
p_1	$\{p_2, p_4, p_9\}$	1		p_7	$\{p_1, p_2, p_3, p_4, p_9, p_{10}, p_{12}\}$	0
p_2	$\{p_4\}$	2		p_8	$\{p_6\}$	1
p_3	$\{p_4, p_{12}\}$	1		p_9	$\{p_4\}$	2
p_4	$\varnothing$	7		p_{10}	$\{p_4\}$	1
p_5	$\{p_6, p_8\}$	0		p_{11}	$\{p_4\}$	0
p_6	$\varnothing$	2		p_{12}	$\varnothing$	2

Tabla 2.3: *S_p y n_p de cada $p \in P$ del ejemplo 2.1.*

para encontrar todos los frentes. A continuación, realizamos el procedimiento para encontrar $\mathcal{F}_2$.

1. Según las líneas 16 y 17 del pseudocódigo, hacemos $i = 1$ para denotar el frente 1, como $\mathcal{F}_1 \neq \varnothing$, entonces inicializamos nuestro conjunto auxiliar $\mathcal{Q} = \varnothing$ y procedemos a analizar cada elemento de $\mathcal{F}_1$ (líneas 18 y 19 del pseudocódigo).

2. Tomemos primero a $p_5 \in \mathcal{F}_1$ y luego, al primer elemento de S_{p_5}, es decir p_6 (líneas 19 y 20).

3. Disminuimos n_{p_6} en una unidad, tal que ahora $n_{p_6} = 1$ (línea 21). Como $n_{p_6} \neq 0$, entonces pasamos a tomar el siguiente elemento de S_{p_5} (líneas 22 y 20 respectivamente).

4. Tomamos $p_8 \in S_{p_5}$ y disminuimos n_{p_8} en una unidad, tal que ahora $n_{p_8} = 0$. Como $n_{p_8} = 0$, entonces hacemos el ranking de n_{p_8} igual a 2 y $\mathcal{Q} = \mathcal{Q} \cup \{p_8\}$ (líneas 23 y 24).

5. Como S_{p_5} no tiene más individuos, entonces tomamos el siguiente elemento de $\mathcal{F}_1$ (línea 19) y relazamos el procedimiento correspondiente a las líneas 20 a 26 para este nuevo individuo.

6. Una vez realizado el procedimiento anterior para cada $p \in \mathcal{F}_1$ incrementamos a i en una unidad y hacemos $\mathcal{F}_2 = \mathcal{Q}$

Terminado el procedimiento anterior, encontrará que $\mathcal{F}_2 = \{p_1, p_3, p_8, p_{10}\}$ y que el ranking p_{rank} de cada elemento de $\mathcal{F}_2$ es ahora 2.

El procedimiento anterior, es repetido nuevamente para encontrar a $\mathcal{F}_3$ usando a $\mathcal{F}_2$ y posteriormente para encontrar a $\mathcal{F}_4$ a partir de $\mathcal{F}_3$. El lector podrá suponer que este procedimiento es innecesario, dado que en la tabla 2.3 se observa que para cada individuo el p_{rank} es igual a $n_p + 1$, donde la jerarquía de cada individuo denota el

frente en el que se encuentra. Sin embargo, esto no siempre sucede así, pues note al individuo p_4.

Por lo tanto, a partir del algoritmo 4 los frentes quedan como:

$$\begin{aligned}\mathcal{F}_1 &= \{p_5, p_7, p_{11}\}\\ \mathcal{F}_2 &= \{p_1, p_3, p_8, p_{10}\}\\ \mathcal{F}_3 &= \{p_2, p_6, p_9, p_{12}\}\\ \mathcal{F}_4 &= \{p_4\}\end{aligned} \tag{2.6}$$

El procedimiento anterior nos permitió establecer una jerarquía en la población P a través de los frentes y/o los rangos de cada elemento. Sin embargo, aún no tenemos una forma de ordenar a los elementos de cada frente de tal manera que podamos decidir cuando uno es mejor que otro, debido a que para cualesquiera $p, q \in \mathcal{F}_i,\ p_{rank} = q_{rank} = i$. Para ello, utilizaremos una estimación de la densidad de la población, que además nos ayudara a mantener uniformente distribuidas a nuestras soluciones en el frente de Pareto aproximado $\mathcal{FP}_{aprox}$.

2.2.2. Estimación de la densidad de población

Una de las cosas que se busca al aproximar el frente de Pareto de un MOP, es que las soluciones en $\mathcal{FP}_{aprox}$ estén uniformemente distribuidas a lo largo de este. Para asegurar esto, es necesario definir una medida de densidad para los elementos de la población que estén en el mismo frente, lo cual además nos permitirá establecer un operador que decida cuando una solución es mejor que otra, prefiriendo a las soluciones que estén en áreas menos densamente pobladas.

2.2.3. Indicador *crowding*

Para aproximar la densidad de población alrededor de un punto i en el frente $\mathcal{F}$, el NSGA-II utiliza una medida de densidad de la población llamada Crowding Distance, a la cual nosotros nos referiremos simplemente como medida de densidad y la denotaremos por d_i.

Intuitivamente, la medida d_i representa la distancia promedio entre los puntos más cercanos alrededor de i, a los cuales denotaremos por $i-1, i+1$, suponiendo que se tiene 2 funciones objetivo como en la figura 2.7. No está de más recalcar que dichos puntos deben estar también en el frente $\mathcal{F}$. Entonces, los individuos que estén en zonas del frente menos densamente pobladas tendrán una medida de densidad alta, lo cual los favorecerá en los operadores de selección, al competir contra soluciones con una medida de densidad baja. Como puede apreciarse también en la figura 2.7, la distancia d_i es utilizada para estimar el perímetro del polígono de líneas punteadas formado con las soluciones $i-1$ e $i+1$, el cual encierra a la solución i. Note que los puntos completamente negros representan los individuos de $\mathcal{F}$ y los blancos en su interior los puntos dominados por los elementos de $\mathcal{F}$. El algoritmo 5 muestra el

Algoritmo 5: asignación_de_medida_de_densidad($\mathcal{F}$) [DPAM02].

Input: Frente $\mathcal{F}$ con l elementos.
Output: Medida de densidad de cada elemento de $\mathcal{F}$.

1 **for** ***each*** $i \in \mathcal{F}$ **do**
2 | $d_i = 0$
3 **end**
4 **for** ***each*** f_k, *con* $k = 1, 2, ..., K$ **do**
5 | $I_k = ordena(I, k)$
6 | $d_{I_1^k} = d_{I_l^k} = \infty$
7 | **for** $j = 2$ *to* $(l-1)$ **do**
8 | | $d_{I_j^k} = d_{I_j^k} + \frac{f_k(I_{j+1}^k) - f_k(I_{j-1}^k)}{f_k^{max} - f_k^{min}}$
9 | **end**
10 **end**
11 **return** $(d_i, \ \forall \ i \in \mathcal{F})$

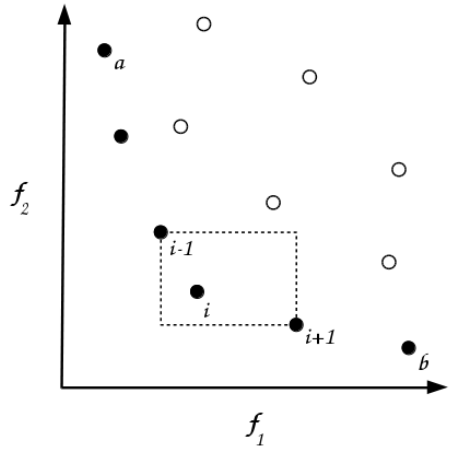


Figura 2.7: *Aproximación de la densidad de población alrededor de un punto i.*

procedimiento necesario para calcular la medida de densidad de cada punto i en el frente $\mathcal{F}$. Dicho procedimiento es explicado a continuación:

- **Paso 1** Calcular el número de soluciones en $\mathcal{F}$ y denotarlo por $l = |\mathcal{F}|$. Además, para cada solución $i \in \mathcal{F}$, hacer $d_i = 0$ (líneas 1 y 2 del pseudocódigo 5).

- **Paso 2** Para cada función objetivo f_k con $k = 1, 2, ..., K$, encontrar su vector de índices $I^k = (I_1^k, I_2^k, ..., I_l^k)$ cuyas entradas son los elementos de $\mathcal{F}$ ordenados de menor a mayor respecto al valor de la función objetivo f_k (líneas 4 y 5).

- **Paso 3** Para cada $k = 1, 2, ..., K$ asignar la mayor distancia a las soluciones en los extremos de I^k, es decir $d_{I_1^k} = d_{I_l^k} = \infty$. Para las demás soluciones $j = 2, ..., (l-1)$ del vector de índices I^k hacer (líneas 7 y 8):

$$d_{I_j^k} = d_{I_j^k} + \frac{f_k(I_{j+1}^k) - f_k(I_{j-1}^k)}{f_k^{max} - f_k^{min}}$$

Donde los parámetros f_k^{max} y f_k^{min} mostrados en la segunda parte de la ecuación del paso 3, así como en la línea 8 del algoritmo 5, son los valores máximos y mínimos (respectivamente) que puede tomar la función objetivo k en su dominio.

Del procedimiento anterior, note que cada función objetivo tendrá su propio vector de índices I^k, donde $d_{I_1^k}$ y $d_{I_l^k}$ denotan las soluciones con el menor y mayor valor de la función objetivo f_k, respectivamente. A estos puntos extremos se les asigna un valor considerado infinito para que siempre sean escogidos, ya que siempre se preferirán a las soluciones con el mayor valor de densidad. En la figura 2.7 los individuos en $\mathcal{F}$ marcados como a y b son precisamente los puntos extremos, los cuales deben ser siempre seleccionados para asegurar que la extensión del frente no se pierda, es por ello que se les asigna un valor de densidad infinito.

En el paso 3, la distancia d_i de cada individuo $i \in \mathcal{F}$, es actualizada cada vez que se pasa a la siguiente función objetivo, ya que la posición o índice de la solución i varía en cada I^k. Una característica importante a notar en el cálculo de la medida de densidad, es que no necesariamente para cada solución i, los vecinos $i-1$ e $i+1$ serán los mismos para cada función objetivo. Este fenómeno puede ocurrir cuando se tienen 3 o más funciones objetivo ($K \geq 3$).

Para que el procedimiento anterior quede un poco más claro, retomaremos el ejemplo 2.1 y calcularemos el valor de densidad de los elementos del frente $\mathcal{F}_2 = \{p_1, p_3, p_8, p_{10}\}$ (mostrado en 2.6), cuyos elementos detallados están en la tabla 2.2.

- **Paso 1** Notamos que $\mathcal{F}_2$ tiene $l = 4$ elementos. Además, hacemos:

$$d_{p_1} = d_{p_3} = d_{p_8} = d_{p_{10}} = 0.$$

- **Paso 2.1** Para f_1, encontramos su vector de índices $I^1 = (p_3, p_{10}, p_1, p_8)$.

- **Paso 3.1** Para los extremos $I_1^1 = p_3$ e $I_4^1 = p_8$ hacemos $d_{I_1^1} = d_{I_4^1} = \infty$ y, para $I_2^1 = p_{10}$, $I_3^1 = p_1$ hacemos:

$$d_{p_{10}} = 0 + \frac{f_1(p_1) - f_1(p_3)}{f_1^{max} - f_1^{min}} = \frac{0.31 - 0.22}{1 - 0.1} = 0.10$$
$$d_{p_1} = 0 + \frac{f_1(p_8) - f_1(p_{10})}{f_1^{max} - f_1^{min}} = \frac{0.79 - 0.27}{1 - 0.1} = 0.58$$

Ahora realizaremos nuevamente los pasos 2 y 3, pero esta vez para f_2.

- **Paso 2.2** Para f_2, encontramos su vector de índices $I^2 = (p_8, p_1, p_{10}, p_3)$.

- **Paso 3.2** Para $I_1^2 = p_8$ e $I_4^2 = p_3$ hacemos $d_{I_1^1} = d_{I_4^1} = \infty$ y, para $I_2^2 = p_1$, $I_3^2 = p_{10}$ hacemos:

$$d_{p_1} = d_{p_1} + \frac{f_1(p_{10}) - f_1(p_8)}{f_1^{max} - f_1^{min}} = 0.58 + \frac{6.93 - 3.97}{60 - 0} = 0.63$$
$$d_{p_{10}} = d_{p_{10}} + \frac{f_1(p_8) - f_1(p_{10})}{f_1^{max} - f_1^{min}} = 0.10 + \frac{7.09 - 6.10}{60 - 0} = 0.12$$

De este modo, la medida de densidad de cada elemento de $\mathcal{F}_2$ queda como:

$$d_{p_1} = 0.63,\ d_{p_3} = \infty,\ d_{p_8} = \infty,\ d_{p_{10}} = 0.12$$

Las medidas de densidad correspondientes a cada elemento de $\mathcal{F}_2$ pueden ser apreciadas en los polígonos de la figura 2.8. Note que los polígonos azules corresponden a los puntos extremos del frente, es decir d_{p_3} y d_{p_8} los cuales se extienden infinitamente. Mientras que los polígonos de color verde correspondientes a d_{p_1}, $d_{p_{10}}$ y tienen un perímetro finito. Más aún, note que el polígono correspondiente a p_{10} es notoriamente más pequeño que el de p_1, lo cual concuerda con sus respectivas medidas de densidad, que son una aproximación del perímetro de sus respectivos polígonos.

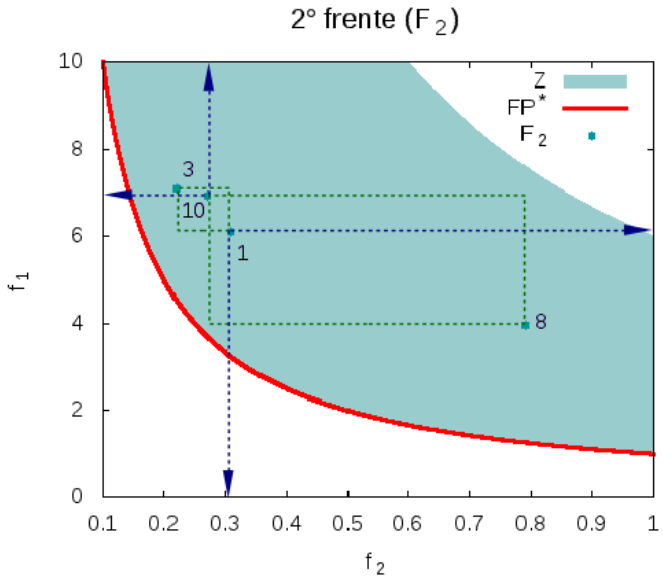


Figura 2.8: *Polígonos de cada solución de* $\mathcal{F}_2$.

Ya que ha quedado claro el concepto de medida de densidad (*Crowding Distance*), es momento de definir el operador de selección binario y el operador de selección de supervivientes, usados por el NSGA-II.

2.2.4. Torneo bajo *crowding*

El Crowded Tournament Selection Operator (CTSO) es un operador binario[5] de selección que realiza una competencia entre 2 soluciones y devuelve al ganador del torneo.

Como vimos en los párrafos anteriores, todo individuo p de una población P tiene 2 atributos:

[5]Se le llama operador binario debido a que solo trabaja con 2 soluciones por vez, es decir, no tiene nada que ver con el tipo de representación.

1. Un *ranking* o clasificación p_{rank} en la población, que denota el frente en el que se encuentra.

2. Una medida de densidad d_p que denota qué tan poblado esta su respectivo frente a su alrededor.

Cada elemento de la población tiene una única medida de densidad que no puede ser compartida con ninguna otra solución. A partir de este hecho y las características anteriores, definimos el siguiente operador.

Definición 2.13 (Crowded Tournament Selection Operator)**.** *Sean $i, j \in P$, la solución i ganara el torneo contra la solución j ($i <_c j$) si alguna de las dos condiciones siguientes se cumple:*

1. *La clasificación o ranking de la solución i es menor que el de j. Esto es, $i_{rank} < j_{rank}$.*

2. *Si sus clasificaciones son iguales pero la solución i tiene una mejor medida de densidad que j. Esto es, $i_{rank} = j_{rank}$ y $d_i > d_j$.*

Notación. *Nos referiremos al Crowded Tournament Selection Operator simplemente como "Operador de Selección Binario" y lo denotaremos por $<_c$.*

Observación. *Note que la relación binaria $<_c$ define un orden parcial estricto sobre el conjunto de soluciones o población P (ver definición 2.6).*

De la observación anterior, como $(P, <_c)$ es un conjunto parcialmente ordenado, entonces dado cualquier par de soluciones $i, j \in P$, solo es posible alguna de las siguientes dos posibilidades:

1. La solución i no gana el torneo contra la solución j, es decir $i <_c j$.

2. La solución j no gana el torneo contra la solución i, es decir $j <_c i$.

La primera condición de la definición 2.13 se asegura de seleccionar la soluciones que se encuentren en un mejor frente. Esto tiene una repercusión importante en la eficiencia del NSGA-II, ya que implica que algunas veces bastara solo con encontrar los distintos frentes para determinar cuándo una solución es mejor que otra. Esto tendrá como consecuencia que no sea necesario calcular las medidas de densidad de cada elemento de uno o varios frentes, e incluso en ocasiones, de todos estos. Si ambas soluciones a competir tienen el mismo ranking y/o se encuentran en el mismo frente, entonces las medidas de densidad de los elementos del frente donde se encuentran serán computadas y se escogerá a la solución que se encuentre en un área menos densamente poblada, es decir, la que tenga la medida de densidad d más alta.

Ya que hemos definido el operador de selección binario es momento de definir el "operador de selección y cruza", así como el "operador de selección de supervivientes".

Algoritmo 6: operador_de_selección_y_cruza(P_t).

Input: P_t
Output: Q_t

$Q_t = \varnothing$
$i = 0$
do
  for *j=1 to 2* **do**
    Escoger 2 individuos de la población de forma aleatoria, $x_1, x_2 \in P_t$
    if $x_1 <_c x_2$ **then**
      $padre\_j = x_1$
    else
      $padre\_j = x_2$
    end
  end
  if *Número aleatorio* $u \leq p_c$ **then**
    Aplicamos el SBX a $padre\_1, padre\_2$ y obtenemos $hijo\_1, hijo\_2$
  else
    Hacemos $hijo\_1 = padre\_1, hijo\_2 = padre\_2$
  end
  $Q_t = Q_t \cup \{hijo\_1, hijo\_2\}$
  $i = i + 2$
while $i < \lambda$
return (Q_t)

2.2.5. Operadores de Selección y Cruza

En la sección 1.4.1 analizamos el operador de selección de padres de nuestro AG mono-objetivo y al igual que en éste, el NSGA-II requiere también de uno para así poder construir una población de descendientes.

Para simplificar las cosas en el NSGA-II, definiremos el operador de selección de padres y el operador de cruza juntos, y lo llamaremos simplemente Operador de Selección y Cruza (OSC).

Dada la generación t (con $1 \leq t \leq (n-1)$) el OSC construye una población de descendientes Q_t a partir de la población de padres P_t. Para ello, el OSC lleva a cabo un torneo binario y así seleccionar a cada uno de los 2 padres. En esencia la forma de selección de padres del NSGA-II es exactamente la misma que la vista en el pseudocódigo del algoritmo 3 de la sección 1.4.1. La diferencia es que el pseudocódigo del algoritmo 3 muestra el proceso para seleccionar j padres llevando a cabo un torneo entre k individuos para escoger a cada uno de los j padres, mientras que el NSGA-II realiza un proceso para seleccionar $j = 2$ padres por medio de un torneo entre $k = 2$ individuos, como puede apreciarse en las líneas 4 a 11 del pseudocódigo del algoritmo 6. El pseudocódigo del algoritmo 3 indica en la línea 5 que debe seleccionarse al individuo mejor adaptado para ser uno de los padres, para el NSGA-II este procedimiento es llevado a cabo en las líneas 6 a 10 del algoritmo 6 por medio del operador se selección binario $<_c$ visto en la definición 2.13.

Una vez que se tienen ambos padres, el siguiente paso es decidir si éstos serán sometidos al operador de cruza binario simulado SBX (visto en la sección 1.4.2) para producir descendencia. Para ello, el algoritmo selecciona un número aleatorio $u \in (0, 1)$ y verifica que se cumpla la condición $u \leq p_c$ (línea 12). Si la condición se cumple, entonces el SBX producirá dos descendientes y éstos se agregaran a la población de hijos (líneas 13 y 17). De lo contrario, el SBX no es aplicado y los padres ingresan directamente a la población de descendientes (líneas 15 y 17). Este procedimiento es llevado a cabo $\lambda/2$ veces (razón por la cual el tamaño de la población λ debe ser un umero par), es decir, hasta que la población de descendientes tenga λ individuos.

Note que el procedimiento anterior sugiere que dada una población P_t de tamaño λ, el OSC producirá una población de descendientes Q_t también de tamaño λ, es decir, mantiene constante el tamaño de la población. El algoritmo 6 muestra el procedimiento del OSC.

2.2.6. Operador de Selección de Supervivientes

El Operador de Selección de Supervivientes (OSS) se encargará de escoger a los individuos mejor adaptados en la generación t, con $t = 1, 2, ..., (G-1)$, de entre la población de padres P_t, y la población de hijos Q_t, para conformar población de padres de la siguiente generación P_{t+1}.

Cabe destacar que el operador de selección es elitista (ver sección 1.4.4), ya que este se apoya en la medida de densidad para seleccionar a los mejores individuos de

Algoritmo 7: operador_de_selección_de_supervivientes(R_t) [DK01].

Input: R_t
Output: P_{t+1}
1 clasificación_veloz_de_elementos_no_dominados(R_t)
2 $P_{t+1} = \varnothing$
3 $i = 1$
4 **while** $(|P_{t+1}| + |\mathcal{F}_i|) < \lambda$ **do**
5 $\quad P_{t+1} = P_{t+1} \cup \mathcal{F}_i$
6 $\quad i = i + 1$
7 **end**
8 **if** $|P_{t+1}| < \lambda$ **then**
9 $\quad$ asignación_de_medida_de_densidad(c)
10 $\quad P_{t+1} = P_{t+1} \cup \{p_i \in \mathcal{F}_i \mid 1 \leq i \leq (\lambda - |P_{t+1}|)\}$
11 **end**
12 **return** (P_{t+1})

entre las poblaciones de padres e hijos.
Notación. *Por R_t nos referiremos al conjunto formado por la población de padres e hijos en la generación t, para $t = 1, 2, ..., G$. Es decir, $R_t = P_t \cup Q_t$.*

El funcionamiento del OSS es mostrado en el algoritmo 7, explicado en el siguiente procedimiento e ilustrado en la figura 2.9.

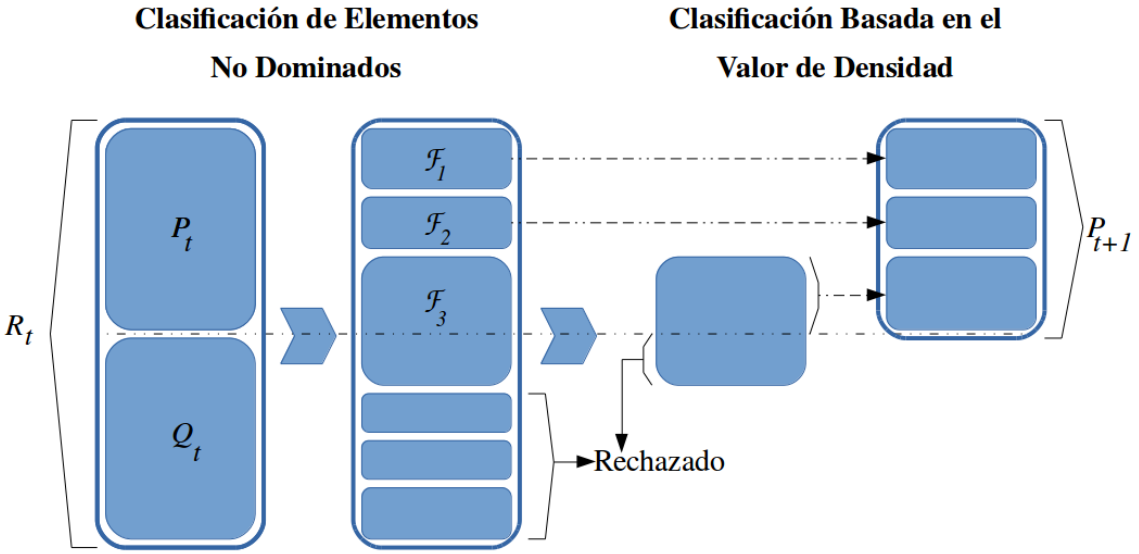


Figura 2.9: *Procedimiento del OSS y/o Algoritmo 7.*

- **Paso 1** Realizar la clasificación veloz de elementos no dominados sobre R_t (línea 1 del algoritmo 7) para encontrar todos sus frentes (ver algoritmo 4).

- **Paso 2** Iniciar siguiente población de padres, $P_{t+1} = \varnothing$, y el índice de los frentes, $i = 1$ (líneas 2 y 3). Mientras que el número de lugares disponibles en P_{t+1} sea mayor que la cantidad de elementos en $\mathcal{F}_i$ ($(|P_{t+1}| + |\mathcal{F}_i|) < \lambda$), agregar $\mathcal{F}_i$ a P_{t+1} y aumentar el índice i ($P_{t+1} = P_{t+1} \cup \mathcal{F}_i$, $i = i + 1$), líneas 4 a 6.

- **Paso 3** Si el último frente seleccionado no pudo ser agregado completamente a P_{t+1} debido a su tamaño (línea 8) entonces, ordenamos los elementos de $\mathcal{F}_i$ en función de la medida de densidad d de cada uno de sus elementos (línea 9, consultar algoritmo 5).

- **Paso 4** Incluir a los primeros $(\lambda - |P_{t+1}|)$ individuos de $\mathcal{F}_i$ a P_{t+1}. Es decir, a los $(\lambda - |P_{t+1}|)$ individuos de $\mathcal{F}_i$ cuya medida de densidad haya sido más alta con respecto a los demás (línea 10).

Note que en el paso 3 del procedimiento anterior solo fue necesario calcular las medidas de densidad de los elementos en el último frente seleccionado, por lo que no fue necesario calcular dichas medidas para todos los elementos de R_t, reduciendo así la complejidad computacional del algoritmo 7.

Para concluir la explicación del OSS, retomemos el ejemplo 2.1. Suponga que tenemos una población de $\lambda = 6$ elementos, luego de la tabla 2.2 suponga que, $P_t = \{p_1, p_2, ..., p_6\}$. Además, suponga que después de llevar a cabo los operadores de recombinación sobre P_t, obtenemos a la población de descendientes $Q_t = \{p_6, p_7, ..., p_{12}\}$ (también de la tabla 2.2). De este modo R_t queda como:

$$R_t = P_t \cup Q_t = \{p_1, p_2, ..., p_{12}\}$$

Después de llevar a cabo la clasificación veloz de elementos no dominados sobre R_t, tenemos que los frentes quedan como (ver sección 2.2.1):

$$\begin{aligned}
\mathcal{F}_1 &= \{p_5, p_7, p_{11}\} \\
\mathcal{F}_2 &= \{p_1, p_3, p_8, p_{10}\} \\
\mathcal{F}_3 &= \{p_2, p_6, p_9, p_{12}\} \\
\mathcal{F}_4 &= \{p_4\}
\end{aligned}$$

Sin embargo, nosotros solo necesitamos 6 elementos para llenar P_{t+1}. Luego, como $\mathcal{F}_1$ tiene solamente 3 individuos, entonces agregamos enteramente a $\mathcal{F}_1$ a P_{t+1}, es decir:

$$P_{t+1} = P_{t+1} \cup \mathcal{F}_1 = \varnothing \cup \{p_5, p_7, p_{11}\} = \{p_5, p_7, p_{11}\}$$

Como $\mathcal{F}_2$ tiene más elementos que lugares disponibles en P_{t+1} (4 y 3 respectivamente), entonces calculamos las medidas de densidad de cada elemento de $\mathcal{F}_2$, de tal forma que obtenemos (ver sección 2.2.3):

$$d_{p_1} = 0.63,\ d_{p_3} = \infty,\ d_{p_8} = \infty,\ d_{p_{10}} = 0.12$$

Por último, seleccionamos a los 3 individuos de $\mathcal{F}_2$ cuya medida de densidad sea más alta con respecto a los demás, y los agregamos a P_{t+1}. Por lo que:

$$P_{t+1} = P_{t+1} \cup \{p_1, p_3, p_8\} = \{p_5, p_7, p_{11}\} \cup \{p_1, p_3, p_8\}$$
$$P_{t+1} = \{p_1, p_3, p_5, p_7, p_8, p_{11}\}$$

Los frentes $\mathcal{F}_3$, $\mathcal{F}_4$ son descartados y de este modo la población de padres de la siguiente generación P_{t+1} es encontrada.

Ya que hemos comprendido bien los conceptos de dominancia y densidad de población, así como sus respectivos operadores, es momento de abordar el algoritmo general del NSGA-II.

2.3. El algoritmo NSGA-II

Algoritmo 8: Algoritmo NSGA-II.

Input: Funciones objetivo $F = (f_1, f_2, ..., f_k)$, restricciones y parámetros.
Output: Conjunto de Pareto evolucionado en n generaciones, P_n.

```
1  Generar población inicial aleatoria P_0 de tamaño λ
2  Calcular F(p) = (f_1(p), f_2(p), ..., f_K(p)) ∀ p ∈ P_0
3  Encontrar frentes de P_0 con
     clasificación_veloz_de_elementos_no_dominados(P_0)
4  Calcular la medida de densidad de cada p ∈ P_0 con
     asignación_de_medida_de_densidad(P_0)
5  for t = 1 to (G − 1) do
6  |   Generar población de descendientes Q_t con
   |     operador_de_selección_y_cruza(P_t)
7  |   Mutar la población de descendientes Q_t
8  |   Calcular F(p) = (f_1(p), f_2(p), ..., f_K(p)) ∀ p ∈ Q_t
9  |   Hacer R_t = P_t ∪ Q_t
10 |   Encontrar P_{t+1} con operador_de_selección_de_supervivientes(R_t)
11 end
12 return (P_n)
```

El NSGA-II sigue básicamente la misma lógica que el AG mono-objetivo explicado en la sección 1.2, incluso utiliza los mismo parámetros mostrados en la tabla 1.2. El NSGA-II incorpora ciertas diferencias debido a sus mejoras y a que el NSGA-II es multi-objetivo. Una de ellas, es que el NSGA-II es elitista y escoge a los mejores individuos de entre la población de padres e hijos. Pues como recordara, para el AG mono-objetivo solo los hijos pasaban a formar parte de la siguiente generación, mientras que los padres eran completamente eliminados. Por ello, el NSGA-II es

capaz de mantener siempre a las mejores soluciones presentes en su población a lo largo de cada generación o iteración.

Sin embargo, el NSGA-II y el AG mono-objetivo que usamos en la sección 1.2 también guardan similitudes fundamentales y que los diferencian de otros AG's. Estas similitudes son el operador de cruza simulado o SBX y el operador de mutación (vistos en las secciones 1.4.2 y 1.4.3 respectivamente), como puede inferirse por la presencia de η y ν en la lista de parámetros. Por ello, cada vez que nos refiramos a la cruza estaremos hablando del SBX. Del mismo modo, cada vez que nos refiramos al operador de mutación, estaremos hablando del visto en la sección 1.4.3.

Sin más preámbulos pasemos ahora a explicar el funcionamiento general del NSGA-II, el cual puede ser apreciado de forma abstracta en el pseudocódigo del algoritmo 8. En la siguiente explicación, cada vez que nos refiramos a alguna de las líneas de un pseudocódigo, nos estaremos refiriendo al algoritmo 8.

A partir de una población de padres P_t, el NSGA-II construye una población de hijos Q_t por medio de la selección y cruza, así como de la mutación (líneas 6 y 7, secciones 2.2.5 y 1.4.3 respectivamente). Sin embargo, en lugar de seleccionar a los individuos de Q_t para generar a P_{t+1}, el NSGA-II combina a la población de padres con la de hijos para formar la población R_t de tamaño 2λ (línea 9). Luego R_t es dividido en frentes para encontrar a los individuos mejor adaptados (posiblemente con la ayuda de la medida de densidad), y así formar a la siguiente población de individuos P_{t+1} (línea 10, sección 2.2.6). El procedimiento anterior se lleva a cabo tantas veces como generaciones n hayan sido especificadas (línea 5).

2.4. Evaluación del desempeño

En la sección 1.5 analizamos algunos problemas de optimización mono-objetivo, en los cuales el optimo global era un valor escalar conocido. Para medir el rendimiento del AG en cada generación, calculábamos la diferencia entre la aptitud del óptimo global y la del individuo mejor adaptado, la cual llamamos error absoluto. Debido a que en la optimización multi-objetivo se trata con un conjunto de soluciones, en lugar de una sola solución, es necesario introducir algunas definiciones que nos permitirán "medir" la distancia entre los frentes; es decir, entre el frente de Pareto real $\mathcal{FP}^*$ y el frente de Pareto aproximado $\mathcal{FP}_{aprox}$ en cada generación. De este modo seremos capaces de decidir cuándo una aproximación a $\mathcal{FP}^*$ es mejor que otra. Debido a esto comenzaremos definiendo que es una medida, y posteriormente daremos algunas de las más usadas en los AG's comenzando por la más conocida, la medida de Hausdorff.

2.4.1. Indicadores

Definición 2.14. *Se le llama espacio métrico a la pareja (X, d), tal X es un conjunto distinto del vacío y $d : X \times X \longrightarrow \mathbb{R}^+ \cup \{0\}$ es una función que cumple que para cualesquiera $x, y, z \in X$:*

I) $d(x, y) \geq 0$

II) $d(x,y) = 0 \Leftrightarrow x = y$

III) $d(x,y) = d(y,x)$

IV) $d(x,y) \leq d(x,z) + d(z,y)$

en dado caso diremos que d es una métrica sobre X.

Definición 2.15. *Sea (X,d) un espacio métrico. Dado $A \subset X$ con $A \neq \varnothing$, definimos:*

$$d(x,A) := inf\{d(x,a) \mid a \in A\}, \text{ para todo } x \in X.$$

Definición 2.16. *Sea (X,d) un espacio métrico, definimos:*

$$\mathcal{H}(X) := \{A \subset X \mid A \neq \varnothing \text{ y } A \text{ es compacto}\}.$$

Definición 2.17. *Sea (X,d) un espacio métrico y $A, B \in \mathcal{H}(X)$, definimos:*

$$\begin{aligned} d(A,B) := sup\{d(a,B) \mid a \in A\} &= sup\{inf\{d(a,b) \mid b \in B\} \mid a \in A\} \\ &= \sup_{a\in A}\{\inf_{b\in B}\{d(a,b)\}\} \end{aligned}$$

Observación. *En general $d(A,B) \neq d(B,A)$.*

Para la observación considere al espacio métrico $(\mathbb{R}, d_u)$ donde d_u es la métrica usual en $\mathbb{R}$. Luego sea $A = \{2\},\ B = \{1,2\} \in \mathcal{H}(\mathbb{R})$, entonces:

$$\begin{aligned} d(A,B) &= sup\{d(a,B) \mid a \in A\} = d(2,B) = inf\{d(2,b) \mid b \in B\} \\ &= inf\{d(2,1), d(2,2)\} = inf\{1,0\} = 0 \\ d(B,A) &= sup\{d(b,A) \mid b \in B\} = sup\{d(1,A), d(2,A)\} = sup\{d(1,2), d(2,2)\} \\ &= sup\{1,0\} = 1 \end{aligned}$$

Luego, $d(A,B) \neq d(B,A)$.

La observación y el ejemplo anterior muestra que por si sola d no es una medida sobre $\mathcal{H}(\mathbb{R})$, por no cumplir la condición III de la definición 2.14. Sin embargo, la siguiente definición introduce una medida en función de d y que además corrige esta falta.

Definición 2.18 (Distancia de Hausdorff)**.** *Sean $A, B \in \mathcal{H}(X)$, definimos la distancia de Hausdorff $d_H : X \times X \longrightarrow \mathbb{R}^+ \cup \{0\}$ como:*

$$d_H(A,B) := max\{d(A,B), d(B,A)\},\ \forall\ A, B \in \mathcal{H}(X).$$

Observación. *d_H es una métrica sobre $\mathcal{H}(X)$.*

Durante mucho tiempo la distancia de Hausdorff fue ampliamente usada para medir la distancia entre frentes y aproximaciones, así como para comparar estas

últimas unas contra otras. Sin embargo, ésta distancia presenta ciertas desventajas significativas que no la hacen ideal para para comparar frentes y/o aproximaciones. La más significativa es que la distancia de Hausdorff penaliza muy severamente los datos atípicos en los conjuntos, provocando que un frente bien aproximado al frente real parezca una mala opción. Para ilustrar mejor esto observe la figura 2.10.

Observación. *Ya sea la distancia de Hausdorff, o alguna otra que definamos en lo subsecuente, siempre trataremos con $\mathcal{H}(\mathbb{R}^n)$ y la distancia euclidiana entre vectores de $\mathbb{R}^n$, a menos que se especifique lo contrario.*

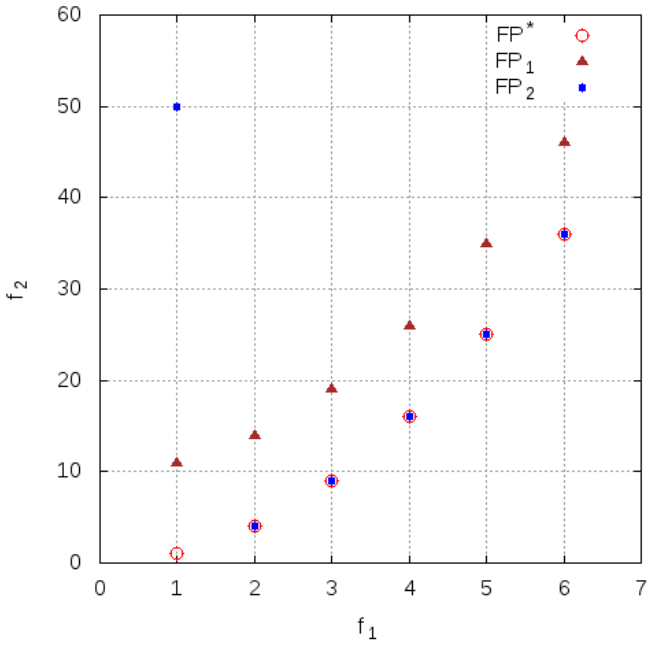


Figura 2.10: *Situación hipotética en la que se tiene el frente de Pareto $\mathcal{FP}^*$ y dos aproximaciones diferentes $\mathcal{FP}_1$ y $\mathcal{FP}_2$.*

En la figura 2.10, se observa claramente que la segunda aproximación $\mathcal{FP}_2$ del frente $\mathcal{FP}^*$ es mejor que la primera $\mathcal{FP}_1$, debido a que $\mathcal{FP}_2$ solo difiere en el primer punto, mientras que $\mathcal{FP}_1$ difiere en todos ellos. Sin embargo, al calcular la distancia de Hausdorff nos encontramos que $d_H(\mathcal{FP}^*, \mathcal{FP}_1) = 10$, mientras que $d_H(\mathcal{FP}^*, \mathcal{FP}_2) = 49$. Es decir, esta sugiere que $\mathcal{FP}_1$ es una mejor aproximación al frente $\mathcal{FP}^*$, lo cual es claramente un error.

Debido a que la distancia de Hausdorff penaliza demasiado a las soluciones atípicas, muchos investigadores han propuesto distintas alternativas. Sin embargo, la mayoría de ellas fueron inspiradas en ésta, por lo que siempre será digna de mención. Dos de ellas son la *Distancia Generacional* y la *Distancia Generacional Invertida* (GD, IGD por sus siglas en ingles), definidas a continuación [SELC12].

Definición 2.19 (Distancia Generacional GD). *Sean $A, B \subset X$ con $A = \{a_1, a_2, ..., a_N\}$ y $B = \{b_1, b_2, ..., b_M\}$ distintos del vacío, definimos la Distancia Generacional GD : $X \times X \longrightarrow \mathbb{R}^+ \cup \{0\}$ como:*

$$GD(A, B) := \frac{1}{N}\left[\sum_{i=1}^{N}\left(\inf_{b\in B} \parallel a_i - b \parallel_q\right)^p\right]^{1/p}.$$

Definición 2.20 (Distancia Generacional Invertida IGD). *Sean $A, B \subset X$ con $A = \{a_1, a_2, ..., a_N\}$ y $B = \{b_1, b_2, ..., b_M\}$ distintos del vacío, definimos la Distancia Generacional Invertida $IGD : X \times X \longrightarrow \mathbb{R}^+ \cup \{0\}$ como:*

$$IGD(A, B) := \frac{1}{M}\left[\sum_{i=1}^{M}\left(\inf_{a\in A} \parallel b_i - a \parallel_q\right)^p\right]^{1/p}.$$

Además de la GD y la IGD, existen algunos otros indicadores de rendimiento utilizados para evaluar los MOEA, tales como: la $\mathcal{S}$ métrica (o hipervolumen), el ϵ-indicador, la proporción de error y la métrica de espaciamiento de Schott, por mencionar los más relevantes [SELC12].

Otro indicador de rendimiento que debe ser mencionado es el Δ_p (delta-p), el cual a través de una modificación sutil al GD y el IGD logra amortizar aún más la penalización de los individuos atípicos en una población. Las modificaciones al GD y el IGD, llamadas GD_p e IGD_p respectivamente, son como sigue:

$$\begin{aligned} GD_p(A, B) &:= \left[\frac{1}{N}\sum_{i=1}^{N}\left(\inf_{b\in B} \parallel a_i - b \parallel_q\right)^p\right]^{1/p}. \\ IGD_p(A, B) &:= \left[\frac{1}{M}\sum_{i=1}^{M}\left(\inf_{a\in A} \parallel b_i - a \parallel_q\right)^p\right]^{1/p}. \end{aligned} \tag{2.7}$$

Note que tanto la GD_p como la IGD_p solo difiere de sus versiones originales en la raíz p-esima de N y M, respectivamente. Sin embargo, esta pequeña modificación tiene consecuencias significativas, como puede verse en [SELC12].

Definición 2.21 (Distancia de Hausdorff promedio Δ_p). *Sean $A, B \subset X$ con $A = \{a_1, a_2, ..., a_N\}$ y $B = \{b_1, b_2, ..., b_M\}$ distintos del vacío. Entonces definimos a $\Delta_p(A, B)$: $X \times X \longrightarrow \mathbb{R}^+ \cup \{0\}$ como:*

$$\Delta_p(A, B) := max(GD_p, IGD_p).$$

Observación. *Para las secciones subsecuentes de este trabajo utilizaremos siempre los indicadores de rendimiento GD, IGD y Δ_p para comparar 2 algoritmos, tratando siempre con subconjuntos finitos de $\mathbb{R}^n$. Además, utilizaremos la norma usual $\parallel \cdot \parallel$ en lugar de la norma $\parallel \cdot \parallel_q$ para calcular la distancia entre vectores o individuos y siempre tomaremos $p = 2$.*

A continuación, daremos algunas propiedades métricas de GD e IGD. Dichas propiedades también se aplican a GD_p e IGD_p, ya que estas solo varían en la raíz p-esima de N y M respectivamente.

Observación. *Para todo $A, B \subset X$ finitos, se cumple que:*

$$IGD(A, B) = GD(B, A).$$

De la observación anterior se sigue que, toda propiedad de GD se aplica a IGD y viceversa (del mismo modo, se cumplen para GD_p e IGD_p).

Observación. *Para todo $A, B \subset X$ finitos, se cumple que:*

$$GD(A, B) \geq 0.$$

A pesar de la observación anterior, GD no cumple la propiedad II de la definición 2.14, ya que para cualesquiera $A, B \subset X$ finito se cumple que:

$$GD(A, B) = 0 \;\Leftrightarrow\; A \subset B \tag{2.8}$$

Por si fuera poco, GD tampoco es simétrica (propiedad III de la definición 2.14) ni cumple la desigualdad del triángulo (propiedad IV)[6].

En resumen, GD (al igual que IGD, GD_p e IGD_p) tiene propiedades métricas muy pobres. Sin embargo, la observación siguiente muestra que Δ_p tiene mejores propiedades métricas que sus indicadores GD_p e IGD_p.

Observación. *Δ_p es una semi-métrica[7] para $1 \leq p < \infty$, y una métrica para $p = \infty$.*

Se rectifica rápidamente que Δ_∞ es una métrica, pues esta coincide con la distancia de Hausdorff, ya que cuando $p = \infty$ (ver definición 2.17):

$$GD_\infty(A, B) = d(A, B) \tag{2.9}$$

luego

$$\begin{aligned} \Delta_\infty(A, B) &= max(GD_\infty(A, B), IGD_\infty(A, B)) \\ &= max(GD_\infty(A, B), GD_\infty(B, A)) = max(d(A, B), d(B, A)) \\ &= d_H(A, B) \end{aligned} \tag{2.10}$$

[6] Una explicación más profunda de estas afirmaciones puede ser encontrada en la cita [SELC12]

[7] Se le llama semi-métrica a las funciones que cumplan las propiedades I a III de la definición 2.14

Para cerrar la sección de *Evaluación del Desempeño*, retomaremos el ejemplo mostrado en la figura 2.10. En dicha figura, claramente se observa que el $\mathcal{FP}_2$ es una mejor aproximación que $\mathcal{FP}_1$. Pero debido a que d_H penaliza muy severamente a los valores atípicos, el valor de este indicador sugiere que $\mathcal{FP}_1$ es una mejor aproximación de $\mathcal{FP}^*$. Sin embargo, para GD (al igual que IGD, GD_p e IGD_p) los resultados muestran que este es un indicador de desempeño más "justo", ya que otorga un valor más congruente con la gráfica del ejemplo, como puede observarse en la tabla 2.4.

Indicador	$\mathcal{FP}_1$	$\mathcal{FP}_2$
$d_H(\mathcal{FP}^*, \bullet)$	10	49
$GD(\mathcal{FP}^*, \bullet)$	3.61	2.47
$IGD(\mathcal{FP}^*, \bullet)$	4.25	0.52

Tabla 2.4: *Indicadores de desempeño. Tanto GD como IGD fueron calculados con la norma usual y* $p = 1$.

2.4.2. Ejemplo numérico

En la sección 2.1.2 analizamos distintos MOPs de prueba y mostramos que no siempre es posible aproximar el frente de Pareto por medio de una búsqueda aleatoria. Por ejemplo, al analizar el MOP denotado por ZDT3, vimos que la función objetivo concentra los puntos del contra-dominio en una región lejana al frente de Pareto, lo que imposibilita la aproximación del frente sin un algoritmo de búsqueda. Por ello, hemos decidido cerrar este capítulo aplicando el NSGA-II a dicho MOP, con el fin de mostrar que el algoritmo es capaz de realizar una buena aproximar el frente de Pareto, tanto en proximidad como en extensión.

A continuación, definiremos nuevamente el MOP denotado por ZDT3 luego, mostraremos la gráfica comparativa entre el frente de Pareto aproximado y el real. Por ultimo, mostraremos la consistencia del algoritmo al graficar 30 ejecuciones usando distintas semillas aleatorias.

MOP denotado por ZDT3:

$$
\begin{aligned}
&\textit{Minimizar} \\
&\quad F = (f_1(x), f_2(x)),\ \textit{donde} \\
&\quad f_1(x) = x_1, \\
&\quad f_2(f_1, g) = g(x)\left[1 - \sqrt{\frac{f_1(x)}{g(x)}} - \frac{f_1(x)}{g(x)} sen(10\pi f_1(x))\right] \qquad (2.11) \\
&\quad y \\
&\quad g(x) = 1 + \frac{9}{n-1}\sum_{i=2}^{n} x_i \\
&\textit{Sujeto a} \quad 0 \leq x_i \leq 1, \quad i = 1, 2, ..., n
\end{aligned}
$$

con $x \in \mathbb{R}^{30}$.

La figura 2.11 muestra la aproximación del $\mathcal{FP}^*$[8] realizada por el NSGA-II utilizando los parametros mostrados en la tabla 2.5.

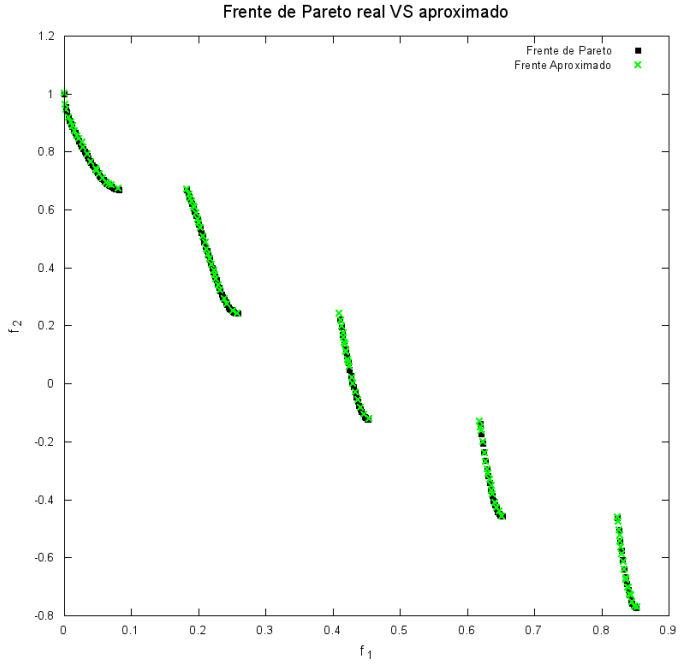


Figura 2.11: *Aproximación del frente de Pareto del MOP ZDT3 usando el NSGA-II.*

Parámetro	G	λ	P_c	P_m	γ	η	ν
Valor	200	100	0.9	0.033	0.123	15	20

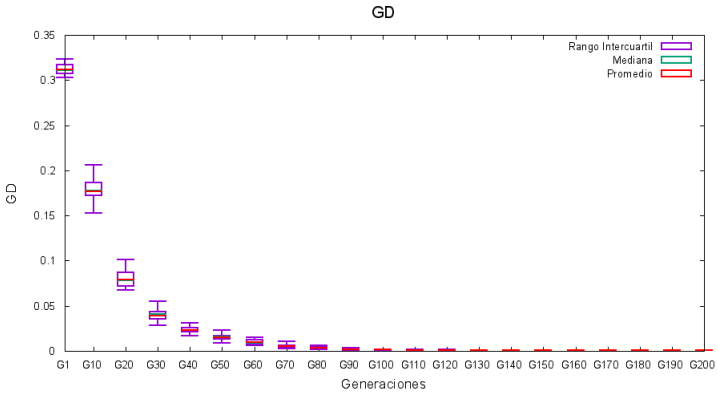
Tabla 2.5: *Parámetros usados en la aproximación de $\mathcal{FP}^*$ del ZDT3 usando el NSGA-II.*

Para mostrar la consistencia del NSGA-II en la aproximación de $\mathcal{FP}^*$, ejecutaremos el algoritmo 30 veces con distintas semillas γ. Además, usaremos los indicadores de desempeño GD, IGD y Δ_p para medir la distancia entre los frentes $\mathcal{FP}^*$ y $\mathcal{FP}_{aprox}$ en cada generación.

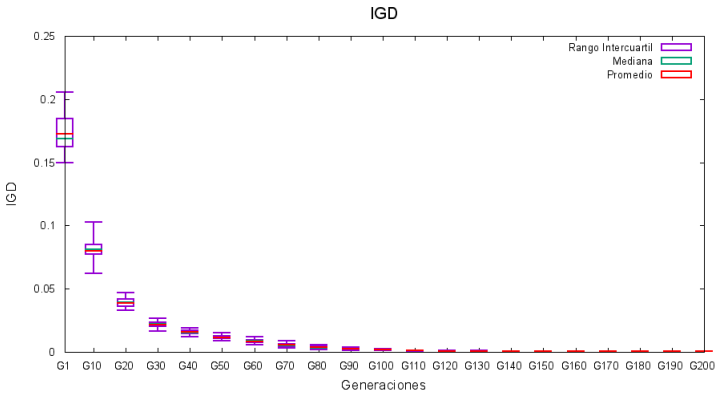
Las gráficas 2.12a, 2.12b y 2.12c muestran el rendimiento del NSGA-II utilizando los indicadores GD, IGD y Δ_p respectivamente a lo largo de las 200 generaciones.

Por último, la gráfica 2.13 muestra el comportamiento del indicador Δ_p de la generación 180 a la 200. En dicha gráfica puede apreciarse que a pesar de que la aproximación a $\mathcal{FP}^*$ es buena, la mejora no es monótona. Ésto se debe al operador de

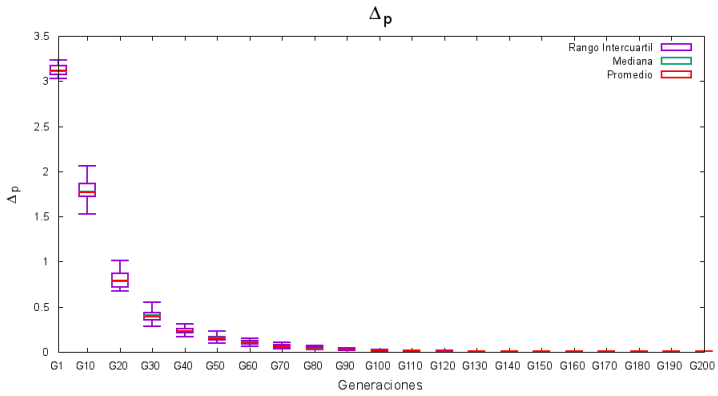
[8]La discretización de $\mathcal{FP}^*$ que utilizamos, puede ser consultada en http://www.cs.cinvestav.mx/~emoobook/apendix-d/data/paretoZDT3.dat



(a) *GD* de las 30 ejecuciones en cada generación.



(b) *IGD* de las 30 ejecuciones en cada generación.



(c) Δ_p de las 30 ejecuciones en cada generación.

Figura 2.12: *Indicadores de desempeño en cada generación.*

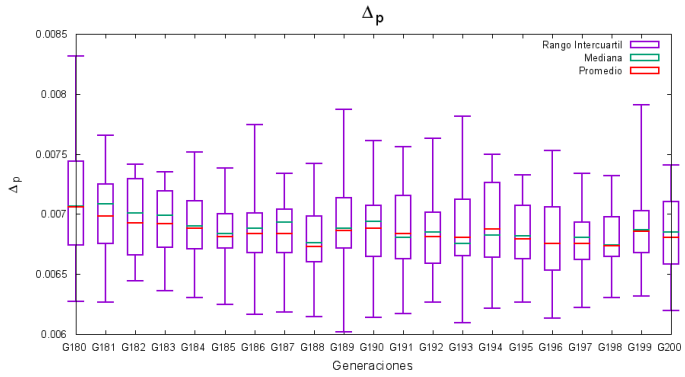


Figura 2.13: *Comportamiento del indicador Δ_p en las últimas 20 generaciones.*

selección de supervivientes ya que, como el lector recordara, éste operador compara dos características del individuo al momento de llevar a cabo la selección *(i)* la jerarquía p_{rank} (la cual denota el frente en el que éste se encuentra) y *(ii)* el valor de la medida de densidad d_p (que denota qué tan densa es la población al rededor del individuo). Una vez que $\mathcal{FP}_{aprox}$ se encuentra cercano a $\mathcal{FP}^*$ los individuos de la población tenderán a estar en el conjunto de elementos no dominados (es decir, $\mathcal{F}_1$), por lo que la selección de los individuos estará determinada enteramente de su valor de densidad d_p, lo cual no tiene relación ni determina la cercanía de $\mathcal{FP}_{aprox}$ a $\mathcal{FP}^*$[9].

[9]Recuerde que el objetivo del operador de selección basado en *crowding* es generar y mantener poblaciones bien distribuidas a lo largo de todo el frente de Pareto.

Capítulo 3

Algoritmo híbrido

3.1. Adaptación del buscador local

En este capítulo propondremos una hibridación entre el NSGA-II y una adaptación multiobjetivo del Método de Descenso con Pendiente Máxima (MDPM), él cual es un algoritmo de búsqueda local basado en gradientes. Primeramente expondremos el MDPM y su extensión para el tratamiento de problemas con dos funciones objetivos. A continuación se presenta la hibridación propuesta y finalmente se analiza la evidencia experimental de la mejora obtenida.

3.1.1. Método de descenso con pendiente máxima

A continuación, enunciaremos algunas definiciones necesarias para esta sección.

Definición 3.1 (Vector Gradiente). *Sea $f : U \subset \mathbb{R}^n \to \mathbb{R}$ una función diferenciable en el conjunto abierto $U \subset \mathbb{R}^n$. Definimos el gradiente de la función f, denotado por ∇f, como el vector cuyas funciones coordenadas son las derivadas parciales de f, es decir:*

$$\nabla f(x) = \left(\frac{\partial f(x)}{\partial x_1}, ..., \frac{\partial f(x)}{\partial x_n} \right) \tag{3.1}$$

donde $x \in U$.

Observación. *En lo subsecuente $\langle \cdot, \cdot \rangle$ y $|| \cdot ||$ denotaran el producto interno y la norma usual respectivamente.*

Definición 3.2 (Derivada Direccional). *Sea $f : U \subset \mathbb{R}^n \to \mathbb{R}$ una función diferenciable en el conjunto abierto $U \subset \mathbb{R}^n$. Definimos la derivada direccional de la función f a lo largo del vector $v \in \mathbb{R}^n$, como:*

$$D_v f = \langle \nabla f(x), v \rangle \tag{3.2}$$

donde $x \in U$.

Observación. *En lo subsecuente supondremos que todos los problemas de optimización con los que tratemos serán de minimización.*

El MDPM (Method of Steepest Descent)[NW06] es un algoritmo de búsqueda iterativo ampliamente usado para resolver problemas de optimización no lineal mono-objetivo. Este algoritmo está catalogado como un "método de búsqueda unidireccional" ya que, dado un vector inicial, digamos x, se determina una *dirección de descenso* $g(x)$. Luego, partiendo desde x se busca un nuevo vector x' avanzando una cantidad α en la dirección de $g(x)$, donde el valor de dicho vector en el contradominio debe de ser menor al de x.

Es decir, analíticamente el vector buscado es de la forma:

$$x' = x + \alpha g(x) \tag{3.3}$$

y debe cumplir que $f(x') \leq f(x)$.

Como puede observarse en la ecuación 3.3, antes de encontrar x' es necesario determinar 2 elementos:

1. La dirección de descenso $g(x)$.

2. El tamaño de paso α.

Comencemos determinando la dirección de descenso $g(x)$. Nos interesa una $g(x)$ tal que, al movernos en dicha dirección una cierta distancia, el valor de f en el nuevo vector x' disminuya. Es decir, requerimos que $g(x)$ sea una "dirección de descenso".

Definición 3.3 (Dirección de Descenso)**.** *Sea $f : U \subset \mathbb{R}^n \to \mathbb{R}$ una función diferenciable en el conjunto abierto $U \subset \mathbb{R}^n$. Diremos que el vector $v \in \mathbb{R}^n$ es una dirección de descenso de la función f, si la derivada direccional de f en la dirección del vector v es no positiva. Es decir:*

$$D_v f = \langle \nabla f(x), v \rangle \leq 0$$

donde $x \in U$.

Como el lector puede intuir, el MDPM requiere de la aproximación del gradiente de la función objetivo, lo cual lo hace computacionalmente costoso. Por ello, nos interesa encontrar una dirección conveniente, es decir, requerimos que la dirección escogida garantice que a través de ésta lograremos el máximo decremento de la función objetivo.

Se verifica fácilmente que $-\nabla f(x)$ es una dirección de descenso para f, ya que:

$$D_{-\nabla f(x)} f = \langle \nabla f(x), -\nabla f(x) \rangle = -\langle \nabla f(x), \nabla f(x) \rangle < 0$$

De hecho, $-\nabla f(x)$ es la dirección de descenso con la que se obtiene el mayor decremento de f. Sin embargo, numéricamente resulta conveniente normalizar dicha dirección, esto se puede ver con el siguiente ejemplo:

Ejemplo 3.1. *Supongamos que se desea minimizar la función de* ***Booth*** *(mostrada en el conjunto de imágenes 3.2) por medio del MDPM.*

$$\begin{aligned} &Minimizar \\ &\quad f(x) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2 \\ &Sujeto\ a \\ &\quad -10 \leq x_i \leq 10, \quad i = 1, 2 \end{aligned} \tag{3.4}$$

donde $x^* = (1, 3)$ *y* $f(x^*) = 0$.

Según lo explicado al inicio de esta sección, lo primero que se requiere para minimizar la función de Booth es seleccionar un punto inicial, digamos $x_0 = (-8.5, -9.7)$. El siguiente paso es encontrar la dirección de descenso, en este caso $-\nabla f(x_0) = (196.6, 203)$. Como puede observarse, la magnitud del vector gradiente es muy grande comparada con la diferencia entre el punto inicial y el óptimo, forzando al método a elegir un tamaño de paso muy pequeño, para compensar, lo cual podría acarrear errores numéricos importantes.

Para evitar el inconveniente mostrado en el ejemplo anterior, es necesario suprimir la magnitud de la dirección de descenso $-\nabla f(x)$. Para ello, basta con normalizar dicho vector, de tal forma que la dirección de descenso quedaría como:

$$g(x) = -\frac{\nabla f(x)}{\|\nabla f(x)\|} \tag{3.5}$$

De este modo, la distancia que avanzaremos en dirección de $g(x)$ dependerá solamente del tamaño de paso α. Así, podemos reescribir la ecuación 3.3 como sigue:

$$x' = x - \alpha \frac{\nabla f(x)}{\|\nabla f(x)\|} \tag{3.6}$$

El normalizar a $-\nabla f(x)$ no perturba de ningún modo la dirección a la que este apunta, ya que la dirección normalizada 3.5 no es más que un múltiplo de la dirección original y el ángulo entre ambas direcciones es 0. De este modo, la dirección normalizada mostrada en la ecuación 3.5 es también una dirección de máximo descenso para la función f.

Siguiendo el algoritmo 3.6, la siguiente tarea es encontrar un tamaño de paso α apropiado.

Por si solo el procedimiento para encontrar α representa un problema no trivial completamente independiente, por lo que existen numerosos métodos para aproximar un α adecuado. En nuestro caso, lo aproximaremos utilizando la regla de Armijo, definida a continuación.

Definición 3.4 (Regla de Armijo [LY08])**.** *Sea* $f : U \subset \mathbb{R}^n \to \mathbb{R}$ *una función diferenciable en el conjunto abierto* $U \subset \mathbb{R}^n$*. Diremos que* $\alpha \in \mathbb{R}^+$ *cumple la regla de Armijo si satisface la siguiente desigualdad:*

$$f(x + \alpha g(x)) \leq c\nabla f(x)^T g(x)\alpha + f(x) \tag{3.7}$$

donde $c \in (0, 1)$*,* $x \in U$ *y* $g(x)$ *es una dirección de descenso.*

Notación. *Usualmente la regla de Armijo es denotada como:*

$$\phi(\alpha) \leq l(\alpha) \tag{3.8}$$

donde $\phi(\alpha) = f(x + \alpha g(x))$, $l(\alpha) = c\nabla f(x)^T g(x)\alpha + f(x)$

Además de facilitar la referencia a alguno (o ambos) lados de la desigualdad de la regla de Armijo, la notación mostrada en 3.8 pretende también evidenciar que la parte derecha de la desigualdad tiene la forma de una recta; siendo $c_1\nabla f(x)^T g(x)$ la pendiente y $f(x)$ la ordenada al origen, como puede observase en la figura 3.1.

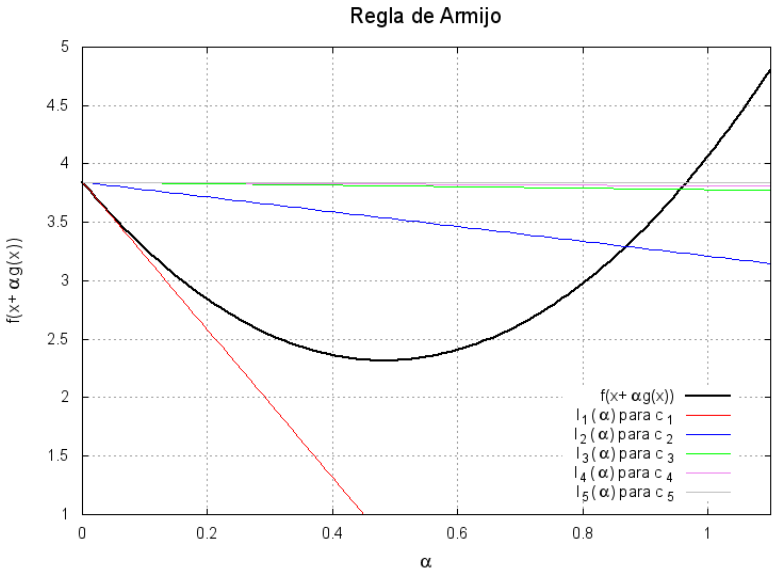


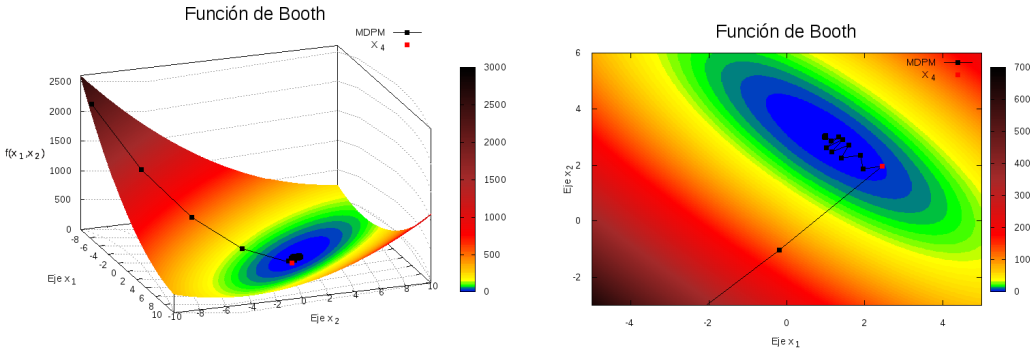
Figura 3.1: *Gráfica de la regla de Armijo aplicada al vector* x_4 *de la función de Booth (ver grupo de figuras 3.2). Donde* $c_1 = 1$, $c_2 = 0.1$, $c_3 = 0.01$, $c_4 = 0.005$ *y* $c_5 = 0$.

La figura 3.1 muestra la regla de Armijo aplicada a la 4^a iteración del MDPM sobre la función de Booth, es decir a x_4 (marcado con rojo en el grupo de figuras 3.2) con el fin de determinar x_5. Esta figura muestra además las rectas $l(\alpha)$ para distintos valores de c, siendo l_1 y l_5 las cotas inferior y superior respectivas a los valores de $c_1 = 1$ y $c_5 = 0$. Note que a medida que c tiende a 0, el intervalo donde α puede ser escogido aumenta, permitiéndole al MDPM dar "pasos" más grandes y llevar a cabo una búsqueda más eficiente, mientras que un c cercano a 1 obliga al MDPM a dar pasos más pequeños y a realizar una búsqueda más exhaustiva. Usualmente c es escogido como 10^{-4}, sin embargo, esto depende de la función que se desee minimizar. Intuitivamente la regla de Armijo permite al usuario seleccionar valores de α grandes con el fin de hacer más eficiente el algoritmo, pero asegurando que dicho valor producirá una mejora en el valor de la función, es decir que el nuevo vector seleccionado cumpla la desigualdad $f(x') \leq f(x)$. En la figura 3.1 puede apreciarse que para valores de c cercanos a 0, la elección de $\alpha > 1$ es posible, sin embargo, esta

elección seria descartada ya que no cumple la desigualdad 3.8, obligando al MDPM a escoger un α más pequeño, como explicaremos a continuación.

El MDPM comienza la búsqueda de una mejora local partiendo de un valor $\alpha = \alpha_{max}$, donde α_{max} es un parámetro establecido por el usuario. Luego, si la condición de Armijo se cumple para dicho α entonces, el vector x' es establecido como $x - \alpha\frac{\nabla f(x)}{\|\nabla f(x)\|}$. De lo contrario el algoritmo reduce $\alpha = \alpha/2$ y lo somete nuevamente a la regla de Armijo. Si el nuevo valor de α no cumple la regla de Armijo, entonces el procedimiento anterior es repetido sucesivamente hasta encontrar un valor α que cumpla dicha regla.

De este modo un nuevo vector x' es hallado, produciendo una mejora en el valor de la función f. El algoritmo antes explicado puede ser aplicado nuevamente al vector x' y así sucesivamente hasta que una mejora local no sea posible. Esto sucederá cuando el modulo del gradiente sea 0, es decir cuando $||\nabla f(x)|| = 0$. En la practica el algoritmo es repetido hasta que $||\nabla f(x)|| < \epsilon$, donde ϵ es un parámetro establecido por el usuario. Un ejemplo práctico del MDPM puede ser apreciado en el grupo de figuras 3.2, donde el MDPM fue ejecutado con los parámetros: $c = 10^{-4}$, $\epsilon = 10^{-4}$, $\alpha_{max} = 4$ y un vector inicial $x = (-8.5, -9.7)$.



(a) Gráfica de la función de Booth y MDPM.

(b) Acercamiento del MDPM para la función de Booth.

Figura 3.2: *Gráficas de la función de Booth y el MDPM.*

3.1.2. Adaptación del MDPM para dos funciones objetivo

El MDPM para un problema de optimización bi-objetivo es básicamente el mismo que el explicado en la sección anterior. Dadas dos funciones objetivo f_1, f_2, un vector inicial $x \in \mathbb{R}^n$, una dirección de descenso $g(x)$ y un tamaño de paso α, calculamos el vector $x' \in \mathbb{R}^n$ con la siguiente expresión:

$$x' = x + \alpha g(x) \tag{3.9}$$

donde debe cumplirse que $f_i(x') \leq f_i(x)$ para cada $i \in \{1, 2\}$.

Ya que es posible que un vector se encuentre muy cercano al frente de Pareto, el procedimiento anterior solo es llevado a cabo si se considera que una mejora local para el vector puede ser alcanzada. Para ello, dicho vector es sometido al criterio mostrado en la ecuación 3.10[1].

Definición 3.5. *Sea $f : \mathbb{R}^n \to \mathbb{R}$, decimos que f es una función de clase C^1 si sus derivadas parciales son continuas.*

Notación. *Sean $f_1, f_2 : \mathbb{R}^n \to \mathbb{R}$ funciones de clase C^1. Denotamos el vector gradiente normalizado de cada función como:*

$$\nabla_i := \frac{\nabla f_i(x)}{\|\nabla f_i(x)\|}$$

para cada $i \in \{1, 2\}$.

Definición 3.6. *Sea $x \in \mathbb{R}^n$ y $f_1, f_2 : \mathbb{R}^n \to \mathbb{R}$ dos funciones de clase C^1 las cuales definen un problema de optimización bi-objetivo. Si la desigualdad:*

$$\left\langle \frac{\nabla f_1(x)}{\|\nabla f_1(x)\|}, \frac{\nabla f_2(x)}{\|\nabla f_2(x)\|} \right\rangle \geq -1 + \epsilon_{tol} \tag{3.10}$$

se cumple entonces, es posible realizar una mejora local sobre el vector x.

El parámetro $0 < \epsilon_{tol} < 0.01$ es especificado por el usuario, y otorga una pequeña tolerancia al criterio.

Ya que hemos establecido un criterio para determinar si una mejora local es (o no) posible, es momento de generalizar otros 2 aspectos fundamentales del algoritmo para un problema de optimización bi-objetivo; La dirección de descenso $g(x)$ y el tamaño de paso α. Del mismo modo que en la sección anterior, procederemos primero con la generalización de $g(x)$.

Definición 3.7 (Cono de Descenso [LCS10])**.** *Sean $f_1, f_2 : \mathbb{R}^n \to \mathbb{R}$ dos funciones de clase C^1, para cada $i \in \{1, 2\}$ definimos:*

$$H^-_{x,i} := \left\{ v \in \mathbb{R}^n | \left\langle \frac{\nabla f_i(x)}{\|\nabla f_i(x)\|}, v \right\rangle \leq 0 \right\}$$

y llamaremos al conjunto $C_x = H^-_{x,1} \cap H^-_{x,2}$ "Cono de Descenso de x".

Definición 3.8 (Dirección de Descenso [LCS10])**.** *Diremos que un vector $v \in \mathbb{R}^n$ es una "dirección de descenso" del vector $x \in \mathbb{R}^n$ si $v \in C_x$.*

La definición 3.8 generaliza para un problema de optimización bi-objetivo el concepto de dirección de descenso visto en la definición 3.3.

[1]El criterio mostrado en la definición 3.6 está inspirado en la condición de optimalidad de Karush-Kkuhn-Tucker (KKT) y fue extraído de la publicación [LCS10].

Proposición 3.1 ([LCS10]). *Sea $x \in \mathbb{R}^n$ y $f_1, f_2 : \mathbb{R}^n \to \mathbb{R}$ dos funciones de clase C^1 las cuales definen un problema de optimización bi-objetivo. Entonces,*

$$\overline{\nabla}_x := -\left(\frac{\nabla f_1(x)}{\|\nabla f_1(x)\|} + \frac{\nabla f_2(x)}{\|\nabla f_2(x)\|}\right) \tag{3.11}$$

es una dirección de descenso de x.

Demostración. Sea $x \in \mathbb{R}^n$, sabemos que $\overline{\nabla}_x$ es una dirección de descenso de x si, $\overline{\nabla}_x \in C_x$. Es decir, si para cada $i \in \{1, 2\}$ se cumple que

$$\left\langle \frac{\nabla f_i(x)}{\|\nabla f_i(x)\|}, \overline{\nabla}_x \right\rangle \leq 0$$

Veamos primero que se cumple para $i = 1$. En efecto

$$\begin{aligned}
\left\langle \frac{\nabla f_1(x)}{\|\nabla f_1(x)\|}, \overline{\nabla}_x \right\rangle &= \left\langle \frac{\nabla f_1(x)}{\|\nabla f_1(x)\|}, -\left(\frac{\nabla f_1(x)}{\|\nabla f_1(x)\|} + \frac{\nabla f_2(x)}{\|\nabla f_2(x)\|}\right) \right\rangle \\
&= -\left\langle \frac{\nabla f_1(x)}{\|\nabla f_1(x)\|}, \left(\frac{\nabla f_1(x)}{\|\nabla f_1(x)\|} + \frac{\nabla f_2(x)}{\|\nabla f_2(x)\|}\right) \right\rangle \\
&= -\left(\left\langle \frac{\nabla f_1(x)}{\|\nabla f_1(x)\|}, \frac{\nabla f_1(x)}{\|\nabla f_1(x)\|} \right\rangle + \left\langle \frac{\nabla f_1(x)}{\|\nabla f_1(x)\|}, \frac{\nabla f_2(x)}{\|\nabla f_2(x)\|} \right\rangle\right) \\
&= -\left(cos(0) + cos(\theta)\right) = -1 - cos(\theta) \\
&\leq 0
\end{aligned}$$

del mismo modo para $i = 2$. Por lo tanto, $\overline{\nabla}_x \in C_x$, es decir, $\overline{\nabla}_x$ es una dirección de descenso de x. □

Del mismo modo que $-\frac{\nabla f(x)}{\|\nabla f(x)\|}$ fue una dirección de descenso para el MDPM original, $\overline{\nabla}_x$ será la dirección de descenso para nuestro MDPM bi-objetivo.

Por lo tanto, podemos reescribir la ecuación 3.9 como sigue:

$$x' = x + \alpha \overline{\nabla}_x \tag{3.12}$$

donde $f_i(x') \leq f_i(x)$ para cada $i \in \{1, 2\}$.

El siguiente paso es generalizar la regla de Armijo, ya que a través de ella garantizaremos que el algoritmo realice una mejora en ambas funciones objetivo.

Definición 3.9 (Regla de Armijo bi-objetivo). *Sea $x \in \mathbb{R}^n$ y $f_1, f_2 : \mathbb{R}^n \to \mathbb{R}$ dos funciones de clase C^1. Diremos que $\alpha \in \mathbb{R}^+$ cumple la regla de Armijo, si para cada $i \in \{1, 2\}$, α satisface la siguiente desigualdad:*

$$f_i(x + \alpha \overline{\nabla}_x) \leq c \nabla f_i(x)^T \overline{\nabla}_x \alpha + f_i(x) \tag{3.13}$$

donde $c \in (0, 1)$.

Notación. *Para cada* $i \in \{1,2\}$*, denotamos la desigualada 3.13 como:*

$$\phi_i(\alpha) \leq l_i(\alpha) \tag{3.14}$$

donde, $\phi_i(\alpha) = f_i(x + \alpha\overline{\nabla}_x)$ *y* $l_i(\alpha) = c\nabla f_i(x)^T\overline{\nabla}_x\alpha + f_i(x)$.

Como puede apreciarse, la regla de Armijo generalizada para dos funciones objetivo al ser comparada con su versión original (definición 3.4) solo difiere en que ahora se pide que la desigualdad 3.13 se satisfaga para ambas funciones objetivo. Por ello, no daremos ninguna otra explicación además de la vista en la sección anterior. El procedimiento para encontrar un α que cumpla la condición 3.9, es también practicante igual al explicado para la condición de Armijo original. Dado el parámetro α_{max} especificado por el usuario, hacemos $\alpha = \alpha_{max}$, si α satisface la regla de Armijo bi-objetivo entonces, α es seleccionado para calcular x' por medio de la ecuación 3.12. De lo contrario, hacemos $\alpha = \alpha/2$ hasta que la condición de Armijo bi-objetivo se cumpla.

Algoritmo 9: mdpm_biobj($x \in \mathbb{R}^n$).

Input: $f_1, f_2 : \mathbb{R}^n \to \mathbb{R},\ x \in \mathbb{R}^n,\ \alpha_{max} \in \mathbb{R}^+,\ c \in (0,1),\ \epsilon_{tol} \in (0, 0.01)$
Output: $x' \in \mathbb{R}^n$

```
Calcular ∇1, ∇2
if (⟨∇1, ∇2⟩ ≥ −1 + ε_tol) then
    Calcular ∇̄_x
    α = α_max
    if (φ_i(α) ≤ l_i(α), para i = 1, 2) then
        x' = x + α∇̄_x
    end
    else
        do
            α = α/2
        while (φ_i(α) > l_i(α), para i = 1, 2)
    end
end
return x'
```

Hemos concluido la extensión del MDPM para un problema de optimización bi-objetivo. El algoritmo 9 muestra en forma abstracta y resumidas cada uno de los pasos del método. Una vez que integremos el MDPM bi-objetivo al NSGA-II, solamente se aplicara una vez el algoritmo 9 a los puntos seleccionados de la población, debido a su alto costo computacional. Sera en la siguiente, y última sección de este trabajo, donde explicaremos la integración del MDPM bi-objetivo al NSGA-II y daremos algunos ejemplos que muestran la ventaja de nuestro algoritmo híbrido, en comparación con el NSGA-II original.

3.2. Hibridación

A continuación presentamos nuestra propuesta de hibridación del NSGA-II descrita mediante el algoritmo 10.

Algoritmo 10: Algoritmo NSGA-II Híbrido.

Input: Parámetros y Funciones objetivo f_1, f_2.
Output: Población evolucionada n generaciones, P_n.

```
1  Generar población inicial aleatoria P_0 de tamaño λ
2  Calcular f_1(p), f_2(p) ∀ p ∈ P_0
3  Encontrar frentes de P_0 con
     clasificación_veloz_de_elementos_no_dominados(P_0)
4  Calcular la medida de densidad de cada p ∈ P_0 con
     asignación_de_medida_de_densidad(P_0)
5  for t = 1 to (n − 1) do
6  |  Generar población de descendientes Q_t con
   |    operador_de_selección_y_cruza(P_t)
7  |  Mutar la población de descendientes Q_t
8  |  Calcular f_1(p), f_2(p) ∀ p ∈ Q_t
9  |  Hacer R_t = P_t ∪ Q_t
10 |  Encontrar P_{t+1} con operador_de_selección_de_supervivientes(R_t)
   |
11 |  if t = ϱ then
12 |  |  for i = 1 to λ do
13 |  |  |  if Rango de p_i es 1 then
14 |  |  |  |  Realizar mejora local del individuo p_i con mdpm_biobj(p_i)
15 |  |  |  end
16 |  |  end
17 |  end
18 end
19 return (P_n)
```

Como era de esperarse, nuestro NSGA-II Híbrido utiliza los mismos parámetros que el NSGA-II original, más los necesarios para que el MDPM bi-objetivo funcione. A éstos se les agrega el parámetro ϱ (línea 11 del algoritmo 10), cuya función es especificar la generación en la cual el MDPM bi-objetivo será ejecutado. Ya que los parámetros del NSGA-II original y el MDPM bi-objetivo fuero explicados con detalle en las secciones 1.3.1 y 3.1.2 respectivamente, nos limitaremos simplemente a listar los parámetros necesarios para nuestro NSGA-II Híbrido, catalogados según su algoritmo de origen:

- **NSGA-II**
  - n Número de generaciones.

- λ Tamaño de la población.
- p_c Probabilidad de cruce.
- p_m Probabilidad de mutación.
- γ Semilla para la generación de números aleatorios.
- η Índice de distribución de cruza.
- ν Índice de distribución de mutación.

▪ **MDPM Bi-Objetivo**

- α_{max} Tamaño de paso máximo.
- c Tolerancia para la regla de Armijo.
- ϵ_{tol} Tolerancia para criterio de inicio.
- ϱ Generación seleccionada para aplicar mejora local.

Del mismo modo, nos limitaremos solamente a explicar las líneas 11 a 17 del algoritmo 10, ya que las demás pueden ser consultadas en las secciones anteriores.

Como puede apreciarse en la línea 11, si el número de la generación t que el NSGA-II esta iterando coincide con el parámetro ϱ entonces, el MDPM bi-objetivo es aplicado a los individuos de la población de supervivientes P_{t+1} cuyo rango sea igual 1 (línea 13), es decir, a todo individuo de $\mathcal{F}_1 \subset P_{t+1}$. Cabe destacar que, ya que el MDPM bi-objetivo trabajo sobre individuos o vectores específicos, y no sobre sobre poblaciones enteras como los AG's entonces, la mejora local será llevada a cabo siempre y cuando el individuo p_i satisfaga el criterio visto en la definición 3.6.

De manera práctica y resumida, una mejora local será llevada a cabo sobre los individuos de la población que el usuario decida y que estén en el frente 1, es decir, a aquellos que individuos que no sean dominados por ningún otro, siempre y cuando dichos individuos no se encuentren demasiado cerca del frente de Pareto.

Con el fin de hacer notar las ventajas que nuestro NSGA-II Híbrido tiene sobre el algoritmo original, a continuación, compararemos algunas funciones de prueba vistas en la sección 2.4, junto con algunas otras que definiremos más adelante.

3.2.1. Resultados numéricos

En esta sección compararemos el rendimiento del NSGA-II original contra nuestro algoritmo Híbrido.

Las funciones de prueba que utilizaremos para comparar ambos algoritmos son la ZDT1, ZDT2, ZDT3, ZDT4 y ZDT6 [CLV06], las cuales pueden ser consultadas en el Apéndice A. Así mismo, los parámetros utilizados en cada una de las funciones de prueba, con las cuales se obtuvieron los resultados y gráficos aquí expuestos, pueden ser consultados la tabla 3.1.

La tabla 3.2 muestra los resultados comparativos entre los algoritmos NSGA-II e Híbrido. La comparación gráfica del rendimiento entre los algoritmos NSGA-II original VS híbrido para los MOPs de prueba ZDT1, ZDT2, ZDT3, ZDT4 y ZDT6,

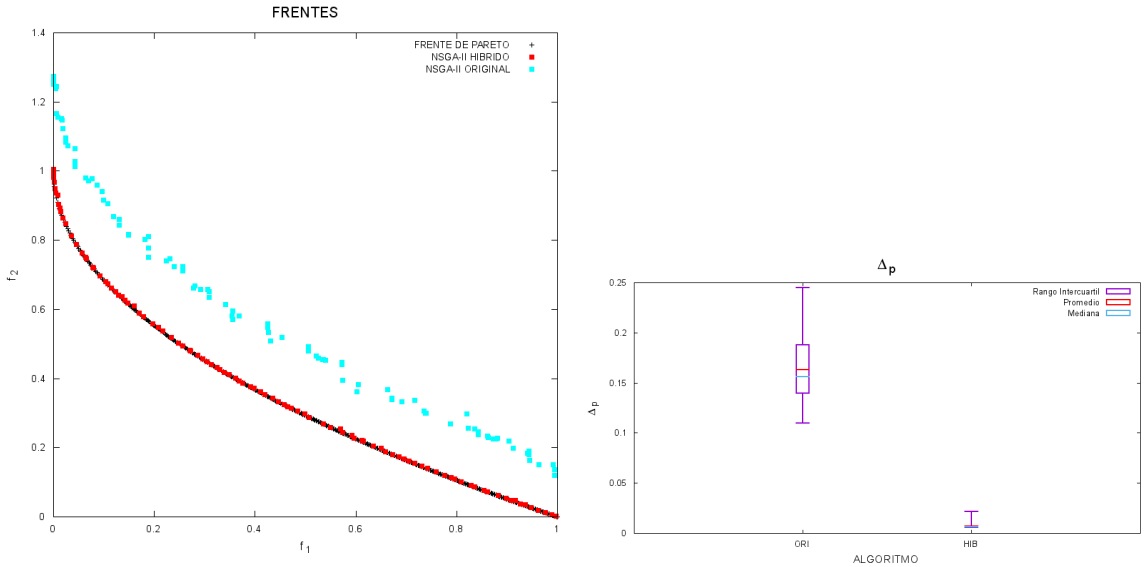
Prob.	Gráfico	γ	NSGA-II Original			MDPM	
ZDT1	3.3b	30 dist.	$G = 50$	$\lambda = 100$	$p_c = 0.9$	$\alpha_{max} = 7$	$c = 0.0001$
	3.3a	0.6492	$p_m = 0.033$	$\eta = 15$	$\nu = 20$	$\epsilon_{tol} = 0.009$	$\varrho = 5$
ZDT2	3.4b	30 dist.	$G = 50$	$\lambda = 100$	$p_c = 0.9$	$\alpha_{max} = 5$	$c = 0.0001$
	3.4a	0.1522	$p_m = 0.033$	$\eta = 15$	$\nu = 20$	$\epsilon_{tol} = 0.009$	$\varrho = 5$
ZDT3	3.5b	30 dist.	$G = 50$	$\lambda = 100$	$p_c = 0.9$	$\alpha_{max} = 30$	$c = 0.0001$
	3.5a	0.8246	$p_m = 0.033$	$\eta = 15$	$\nu = 20$	$\epsilon_{tol} = 0.009$	$\varrho = 5$
ZDT4	3.6b	30 dist.	$G = 120$	$\lambda = 100$	$p_c = 0.9$	$\alpha_{max} = 0.4$	$c = 0.0001$
	3.6a	0.7076	$p_m = 0.1$	$\eta = 15$	$\nu = 20$	$\epsilon_{tol} = 0.009$	$\varrho = 4$
ZDT6	3.7b	30 dist.	$G = 200$	$\lambda = 100$	$p_c = 0.9$	$\alpha_{max} = 0.4$	$c = 0.0001$
	3.7a	0.7953	$p_m = 0.1$	$\eta = 15$	$\nu = 20$	$\epsilon_{tol} = 0.009$	$\varrho = 68$

Tabla 3.1: *Parámetros usados en la ejecución de los algoritmos NSGA-II e Híbrido para aproximar los frentes de Pareto de las funciones de prueba.*

puede ser consultada en los grupos de figuras 3.3, 3.4, 3.5, 3.6 y 3.7 respectivamente. Las columnas 3, 4 y 5 de la tabla 3.2 corresponden al promedio los indicadores de rendimiento *GD*, *IGD* y Δ_p respectivamente, de 30 ejecuciones de los algoritmos con distintas semillas aleatorias. La columna 6 corresponde al numero total de evaluaciones de las funciones objetivo realizadas por cada algoritmo. Por último, las columnas 7 y 8 corresponden al porcentaje de evaluaciones hechas por el NSGA-II y el MDPM respectivamente. Como puede apreciarse en los grupos de figuras 3.3, 3.4, 3.5, 3.6 y 3.7 correspondientes a los MOPs de prueba ZDT1, ZDT2, ZDT3, ZDT4 y ZDT6 respectivamente, nuestro algoritmo híbrido probo ser eficaz para cada una de ellos. Más aún, puede observarse en las columnas 7 y 8 de la tabla 3.2 que al destinar un porcentaje de las evaluaciones de la función objetivo al MDPM, nuestro algoritmo híbrido logro una diferencia significativa en comparación al NSGA-II original. Tal porcentaje representa el promedio de haber ejecutado ambos algoritmos bajo condiciones iguales pero usando 30 semillas aleatorias distintas. Cabe destacar que en todos los casos, el porcentaje correspondiente al MDPM nunca superó el 25 % de las evaluaciones totales de la función objetivo, y en particular para el MOP ZDT4 solo uso el 1 %. Utilizando los mismos MOPs de prueba y por medio de otro experimento, en la siguiente y última sección de este trabajo analizaremos más extensamente la relación costo-beneficio que existe entre el porcentaje usado por el MDPM y el NSGA-II.

Problema	Algoritmo	GD Promedio	IGD Promedio	Δ_p Promedio	Num. de eval. de F	Promedio de eval. de F	
						NSGA-II	MDPM
ZDT1	Original	1.6330E-02	6.3754E-03	1.6330E-01	5,000	100 %	0 %
	Híbrido	**3.0877E-04**	**3.0379E-04**	**6.7929E-03**		81.26 %	19.80 %
ZDT2	Original	3.2118E-02	1.5464E-02	3.5125E-01		100 %	0 %
	Híbrido	**1.4775E-04**	**6.2794E-03**	**1.4041E-01**		91.93 %	8.97 %
ZDT3	Original	1.4775E-02	1.1737E-02	1.4919E-01		100 %	0 %
	Híbrido	**1.4401E-03**	**2.2126E-03**	**2.5804E-02**		76.06 %	24.92 %
ZDT4	Original	4.0554E-02	3.0939E-02	5.1438E-01	12,000	100 %	0 %
	Híbrido	**2.2073E-02**	**1.5760E-02**	**2.5902E-01**		99.22 %	1.23 %
ZDT6	Original	1.6526E-03	4.1528E-04	2.2715E-02	20,000	100 %	0 %
	Híbrido	**1.8960E-04**	**7.9227E-05**	**4.3336E-03**		95.85 %	4.39 %

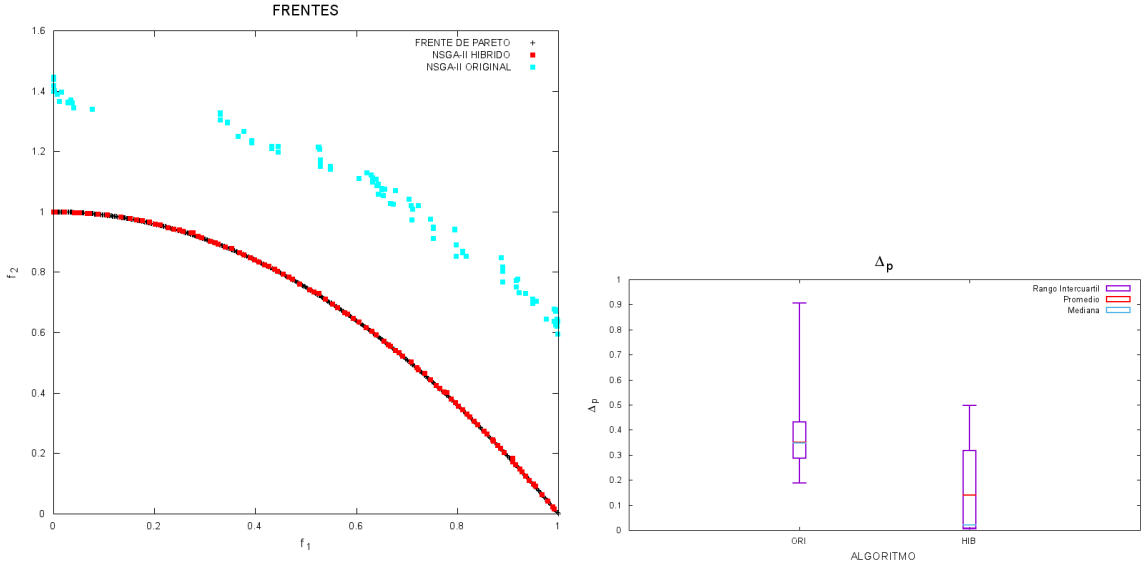
Tabla 3.2: *Tabla comparativa NSGA-II Original vs. Híbrido. Los mejores valores de los indicadores usados se resaltan para cada algoritmo y problema de prueba.*



(a) Frentes del ZDT1 con $\gamma = 0.6492$.

(b) Indicador Δ_p, con $p = 2$.

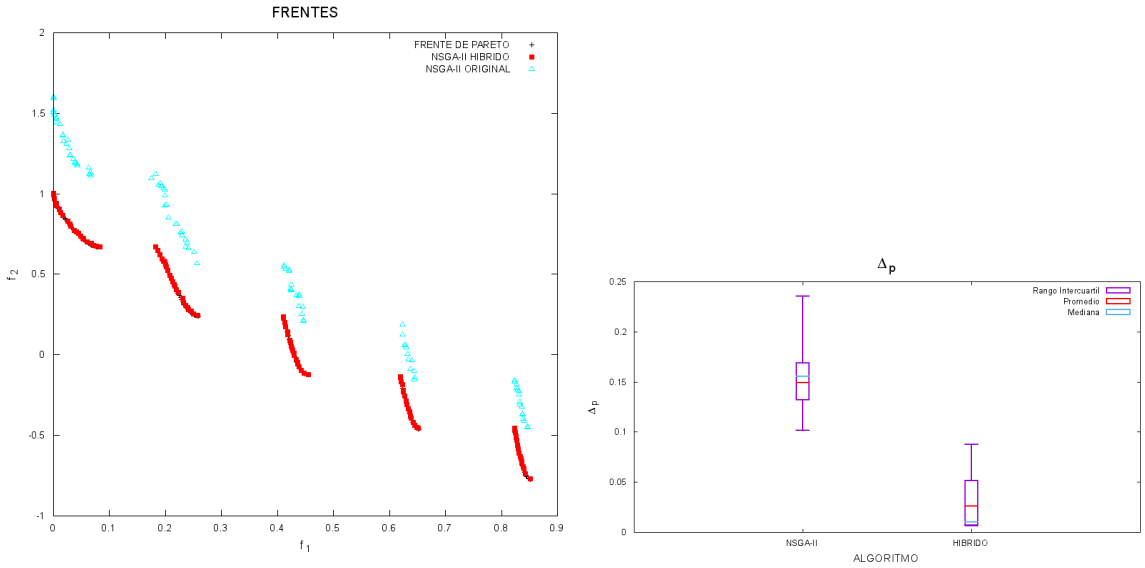
Figura 3.3: *Comparativo del Algoritmo NSGA-II Original vs. Híbrido de 30 ejecuciones usando distintas semillas γ y la función de prueba ZDT1. En la figura (a) se muestra que la aproximación del algoritmo híbrido al frente de Pareto (cuadros rojos) fue mejor que la del NSGA-II (cuadros cyan), al llevar a cabo 5,000 evaluaciones de la función objetivo. La figura (b) muestra que la ventaja del algoritmo híbrido sobre el original es consistente para distintos valores de γ.*



(a) Frentes del ZDT2 con $\gamma = 0.1522$.

(b) Indicador Δ_p, con $p = 2$.

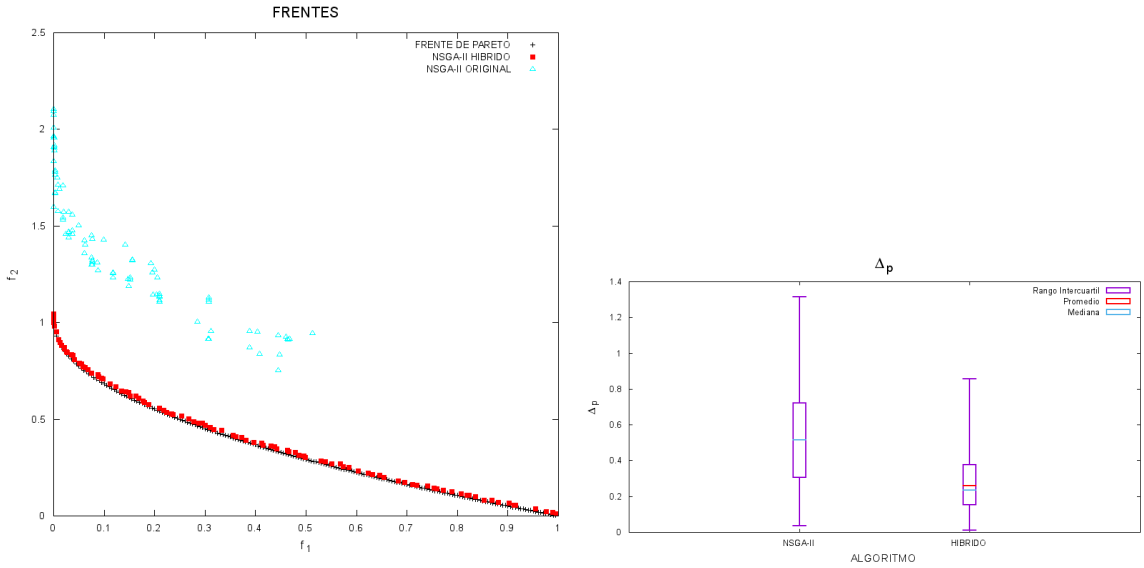
Figura 3.4: *Comparativo del Algoritmo NSGA-II Original vs. Híbrido de 30 ejecuciones usando distintas semillas γ y la función de prueba ZDT2. En la figura (a) se muestra que la aproximación del algoritmo híbrido al frente de Pareto (cuadros rojos) fue mejor que la del NSGA-II (cuadros cyan), al llevar a cabo 5,000 evaluaciones de la función objetivo. La figura (b) muestra que en promedio el algoritmo híbrido fue mejor que el original con respecto al indicador Δ_p para distintos valores de γ.*



(a) Frentes del ZDT3 con $\gamma = 0.8246$.

(b) Indicador Δ_p, con $p = 2$.

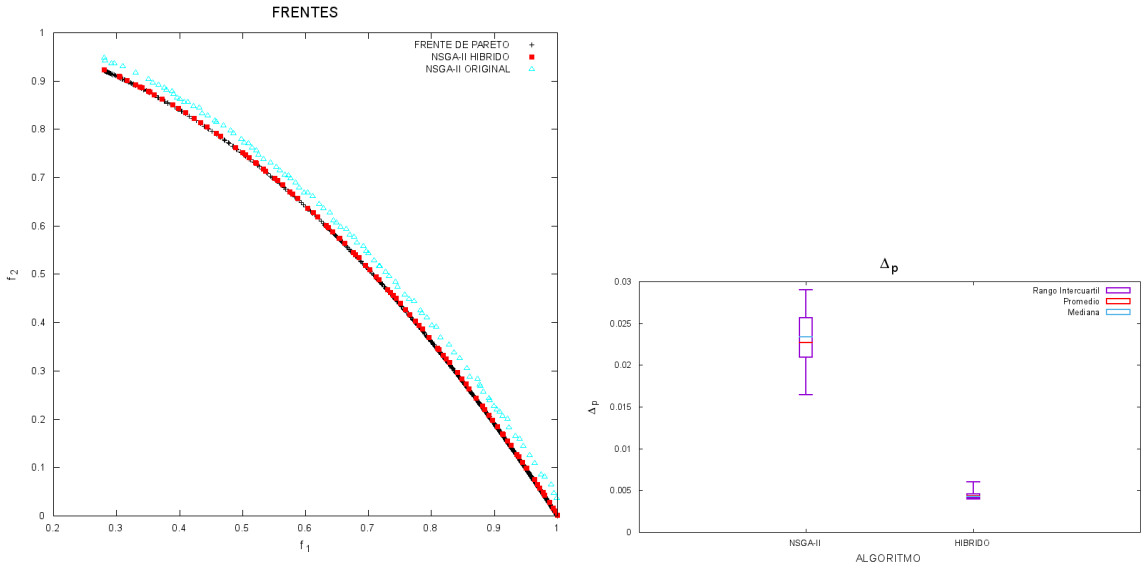
Figura 3.5: *Comparativo del Algoritmo NSGA-II Original vs. Híbrido de 30 ejecuciones usando distintas semillas γ y la función de prueba ZDT3. En la figura (a) se muestra que la aproximación del algoritmo híbrido al frente de Pareto (cuadros rojos) fue mejor que la del NSGA-II (cuadros cyan), al llevar a cabo 5,000 evaluaciones de la función objetivo. La figura (b) muestra que la ventaja del algoritmo híbrido sobre el original es consistente para distintos valores de γ.*



(a) Frentes del ZDT4 con $\gamma = 0.7076$.

(b) Indicador Δ_p, con $p = 2$.

Figura 3.6: *Comparativo del Algoritmo NSGA-II Original vs. Híbrido de 30 ejecuciones usando distintas semillas γ y la función de prueba ZDT4. En la figura (a) se muestra que la aproximación del algoritmo híbrido al frente de Pareto (cuadros rojos) fue mejor que la del NSGA-II (cuadros cyan), al llevar a cabo 12,000 evaluaciones de la función objetivo. La figura (b) muestra que en promedio el algoritmo híbrido fue mejor que el original con respecto al indicador Δ_p para distintos valores de γ.*



(a) Frentes del ZDT6 con $\gamma = 0.7953$.

(b) Indicador Δ_p, con $p = 2$.

Figura 3.7: *Comparativo del Algoritmo NSGA-II Original vs. Híbrido de 30 ejecuciones usando distintas semillas γ y la función de prueba ZDT6. En la figura (a) se muestra que la aproximación del algoritmo híbrido al frente de Pareto (cuadros rojos) fue mejor que la del NSGA-II (cuadros cyan), al llevar a cabo 20,000 evaluaciones de la función objetivo. La figura (b) muestra que la ventaja del algoritmo híbrido sobre el original es consistente para distintos valores de γ.*

3.2.2. Estimación de la eficiencia

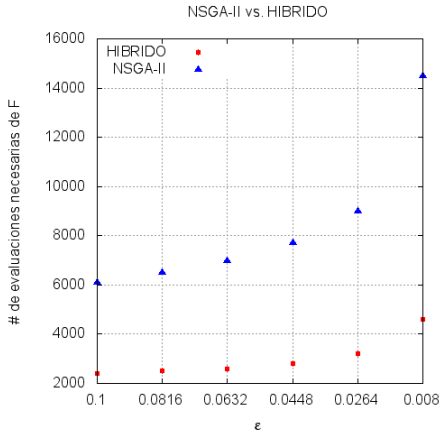
En la sección anterior comparamos nuestro AG híbrido contra el NSGA-II original por medio de las funciones de prueba mostradas en el apéndice A. En dicha comparativa no solo mostramos la eficacia de nuestro algoritmo híbrido en la aproximación de los frentes de Pareto, también hicimos notar que para un número especifico de evaluaciones de la función objetivo, en general nuestro algoritmo realizo mejores aproximaciones al frente de Pareto. En esta sección, y última de este trabajo, compararemos la eficiencia de nuestro algoritmo Híbrido contra la del NSGA-II original. Para ello, analizaremos el numero de evaluaciones de la función objetivo necesarias para que cada algoritmo obtenga un frente de Pareto aproximado $\mathcal{FP}_{aprox}$ que se encuentren a una distancia ϵ del frente real $\mathcal{FP}^*$ bajo la métrica Δ_p, con $p = 2$. El análisis de ambos algoritmos se llevara a cabo por medio de las funciones de prueba definidas en el Apéndice A, así como de seis distancias ϵ distintas. Del mismo modo, los parámetros usados en los algoritmos fueron los mismos que los mostrados en la sección anterior en la tabla 3.1, con excepción del parámetro G ya que para este experimento, G estaba en función de ϵ. Una vez más, los datos obtenidos en este experimento son el resultado de 30 ejecuciones de los algoritmos bajo idénticas condiciones y con distintas semillas aleatorias, con el objetivo de mostrar la consistencia de los resultados. Dichos resultados obtenidos en éste experimento pueden ser apreciados en el grupo de figuras 3.8 y la tabla 3.3.

ϵ	Problema				
	ZDT1	ZDT2	ZDT3	ZDT4	ZDT6
0.1	**60.66 %**	**19.23 %**	**54.10 %**	**12.05 %**	**35.00 %**
0.0816	**61.54 %**	**17.86 %**	**52.31 %**	**11.63 %**	**37.84 %**
0.0632	**62.86 %**	**18.68 %**	**46.48 %**	**12.64 %**	**40.13 %**
0.0448	**63.64 %**	**20.59 %**	**44.87 %**	**14.80 %**	**43.53 %**
0.0264	**64.44 %**	**23.97 %**	**45.05 %**	**11.59 %**	**44.79 %**
0.008	**68.28 %**	**23.27 %**	**43.75 %**	**0.43 %**	**53.90 %**

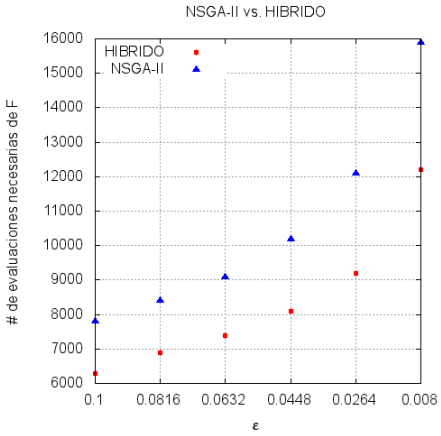
Tabla 3.3: *Los porcentajes indican la reducción en el numero de evaluación de la función objetivo que el algoritmo Híbrido requirió, para aproximar un frente que se encontrase a no más de una distancia ϵ del frente real, en comparación con el algoritmo NSGA-II original.*

La tabla 3.3 muestra los porcentajes correspondientes a la mejora en el rendimiento que el algoritmo Híbrido mostró en comparación con el NSGA-II original, note que para los ϵ utilizados el algoritmo Híbrido mostro siempre una mejora en la eficiencia para todas las funciones de prueba.

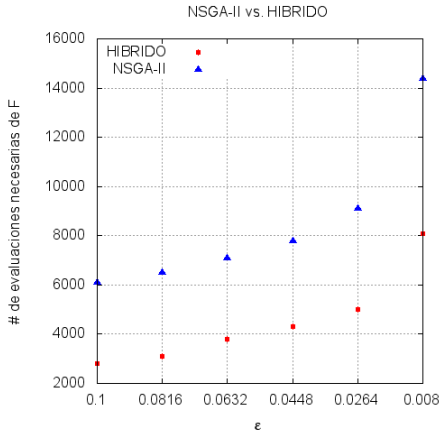
Del grupo de figuras 3.8 y la tabla 3.3, se observa que para la función nombrada ZDT1 la mejora en la eficiencia fue siempre de al menos 60 %, llegando a alcanzar hasta un 68 % en la aproximación más precisa. Para la función nombrada ZDT3 el algoritmo Híbrido logro reducir a la mitad el numero de evaluaciones en los primer



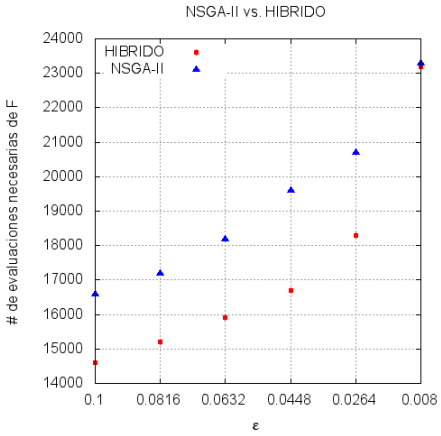
(a) ZDT1.



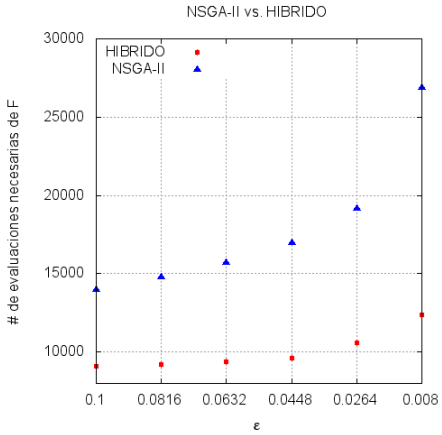
(b) ZDT2.



(c) ZDT3.



(d) ZDT4.



(e) ZDT6.

Figura 3.8: *Comparativo entre algoritmos NSGA-II e Híbrido para distintos ϵ.*

dos ϵ, mostrando siempre una mejora de al menos un 43 %. Lo mismo ocurre para la función nombrada ZDT6, donde para $\epsilon = 0.008$ el numero de evaluaciones se redujo a la mitad, siendo la mejora de al menos 35 % para los demas ϵ. La función nombrada ZDT2 mostró una mejora mínima del 17 % y una máxima de casi 24 %. Por último, la función nombrada ZDT4 mostró una mejora máxima de 14.8 % y una mínima de 0.43 %.

Capítulo 4

Conclusiones

A lo largo de la historia, la naturaleza siempre nos ha brindado la mejor fuente de inspiración en la búsqueda de soluciones a problemas. La ciencia de la computación ha sido una de las disciplinas beneficiadas de ello diseñando los algoritmos genéticos.

Este trabajo se centró en el NSGA-II. Los objetivos que se cumplieron fueron los siguientes: *i)* analizar dicho algoritmo de forma general, así como cada uno de sus operadores, para así lograr un entendimiento basto de su funcionamiento, y *ii)* proponer un método de hibridación basado en gradientes con el objetivo de mejorar la eficiencia del algoritmo. En este trabajo se mostró mediante experimentos numéricos, usando un conjunto de problemas de prueba estándar (ZDT) con características diversas, que el algoritmo híbrido, aquí propuesto, fue eficaz para obtener la aproximación de los conjuntos diversos de soluciones; e incluso resultó ser más eficiente en comparación con el NSGA-II. A partir del análisis realizado al NSGA-II en este trabajo, y de los resultados obtenidos por nuestro algoritmo híbrido, nos es posible concluir dos observaciones fundamentales: *i)* la convergencia del NSGA-II está limitada por el mecanismo de selección basado en el *crowding*, *ii)* sí es posible robustecer un algoritmo estocástico por medio de la incorporación de métodos exactos. Las observaciones anteriores dejan distintas líneas de investigación abiertas. Como trabajo futuro, podría buscarse mejorar el operador de selección del NSGA-II para que éste sea capaz de mantener posibles mejoras de ciertos individuos en cada generación; haciendo así decreciente la medida de la distancia al frente de Pareto conforme transcurren las generaciones. De hacerse lo anterior, la experiencia adquirida en este trabajo nos indica que es posible aplicar el MDPM también en el refinamiento y mejora de las soluciones finales.

Apéndices

Apéndice A

Funciones de prueba biobjetivo

Este apéndice contiene los problemas de optimización bi-objetvo de prueba usados en este trabajo [CLV06]. En todos ellos el objetivo es minimizar las funciones objetivo. Al final de la definición de las función de prueba, se encuentra la representación gráfica del frente de Pareto $\mathcal{FP}^*$ para cada una de ellas, cuyos archivos de datos fueron descargados de la siguiente pagina:

https://www.cs.cinvestav.mx/~emoobook/

ZDT1

$$F = (f_1(x), f_2(x)),\ \textit{donde}$$

$$f_1(x) = x_1,$$
$$f_2(f_1, g) = g(x)\left(1 - \sqrt{\frac{f_1(x)}{g(x)}}\right)$$

y

$$g(x) = 1 + \frac{9}{n-1}\sum_{i=2}^{n} x_i$$

con $0 \leq x_i \leq 1$ para $i \in \{1, ..., 30\}$, $x = (x_1, x_2, ..., x_n)$ y $n = 30$. La representación gráfica del frente de Pareto $\mathcal{FP}^*$ es mostrada en la figura A.1a.

ZDT2

$$F = (f_1(x), f_2(x)),\ donde$$

$$f_1(x) = x_1,$$
$$f_2(f_1, g) = g(x)\left[1 - \left(\frac{f_1(x)}{g(x)}\right)^2\right]$$
$$y$$
$$g(x) = 1 + \frac{9}{n-1}\sum_{i=2}^{n} x_i$$

con $0 \leq x_i \leq 1$ para $i \in \{1, ..., 30\}$, $x = (x_1, x_2, ..., x_n)$ y $n = 30$. La representación gráfica del frente de Pareto $\mathcal{FP}^*$ es mostrada en la figura A.1b.

ZDT3

$$F = (f_1(x), f_2(x)),\ donde$$

$$f_1(x) = x_1,$$
$$f_2(f_1, g) = g(x)\left(1 - \sqrt{\frac{f_1(x)}{g(x)}} - \frac{f_1(x)}{g(x)} sen(10\pi f_1(x))\right)$$
$$y$$
$$g(x) = 1 + \frac{9}{n-1}\sum_{i=2}^{n} x_i$$

con $0 \leq x_i \leq 1$ para $i \in \{1, ..., 30\}$, $x = (x_1, x_2, ..., x_n)$ y $n = 30$. La representación gráfica del frente de Pareto $\mathcal{FP}^*$ es mostrada en la figura A.1c.

ZDT4

$$wF = (f_1(x), f_2(x)),\ donde$$

$$f_1(x) = x_1,$$
$$f_2(f_1, g) = g(x)\left(1 - \sqrt{\frac{f_1(x)}{g(x)}}\right)$$
$$y$$
$$g(x) = 1 + 10(n-1) + \sum_{i=2}^{n}(x_i^2 - 10cos(4\pi x_i))$$

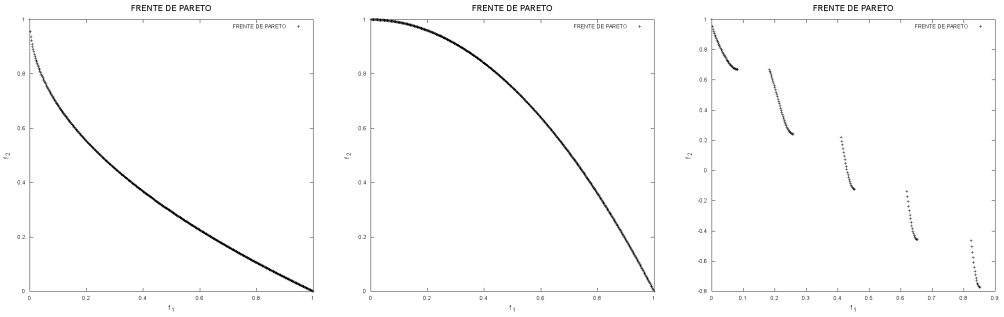
con $0 \leq x_1 \leq 1$ y $-5 \leq x_i \leq 5$ para $i \in \{2, ..., 10\}$, $x = (x_1, x_2, ..., x_n)$ y $n = 10$. La representación gráfica del frente de Pareto $\mathcal{FP}^*$ es mostrada en la figura A.1d.

ZDT6

$$wF = (f_1(x), f_2(x)),\ donde$$

$$f_1(x) = 1 - e^{-4x_1} sen^6(6\pi x_1),$$
$$f_2(f_1, g) = g(x)\left[1 - \left(\frac{f_1(x)}{g(x)}\right)^2\right]$$
$$y$$
$$g(x) = 1 + 9\left(\frac{\sum_{i=2}^{n} x_i}{9}\right)^{1/4}$$

con $0 \leq x_i \leq 1$ para $i \in \{1, ..., 10\}$, $x = (x_1, x_2, ..., x_n)$ y $n = 10$. La representación gráfica del frente de Pareto $\mathcal{FP}^*$ es mostrada en la figura A.1e.



(a) $\mathcal{FP}^*$ de ZDT1. **(b)** $\mathcal{FP}^*$ de ZDT2. **(c)** $\mathcal{FP}^*$ de ZDT3.

(d) $\mathcal{FP}^*$ de ZDT4. **(e)** $\mathcal{FP}^*$ de ZDT6.

Figura A.1: *Representación gráfica del* $\mathcal{FP}^*$.

Índice de figuras

1.1. Problema del viajero. 4
1.2. Comportamiento de un AG como buscador del óptimo global. 8
1.3. Cruza binaria basada en un punto. 14
1.4. Distribución de probabilidad del SBX. 16
1.5. Distribución de probabilidad del SBX para 2 padres. 17
1.6. Distribución de probabilidad del operador de mutación. 20
1.7. Aproximación de un AG a un óptimo local. 22
1.8. Evolución de la población desde un óptimo local hasta alcanzar el global. 23
1.9. Función de Beale. 27
1.10. Aproximación del optimo global en la función de Beale. 28
1.11. Error absoluto para la función de Beale. 28
1.12. Análisis del error absoluto para la función de Beale. 29
1.13. Función de Rastrigin. 30
1.14. Aproximación del optimo global en la función de Rastrigin. 31
1.15. Segunda aproximación del optimo global en la función de Rastrigin. . . . 32
1.16. Análisis del error absoluto para la función de Rastrigin. 33
1.17. Función de Rosenbrock. 34
1.18. Análisis del error absoluto para la función de Rosenbrock. 34

2.1. Ejemplo básico de un MOP. 38
2.2. Frente de Pareto correspondiente al MOP BINH1. 42
2.3. Frente de Pareto del MOP BINH1 graficado en el dominio de la función objetivo. 43
2.4. Frente de Pareto correspondiente al MOP DEB2. 44
2.5. Frente de Pareto correspondiente al MOP ZDT3. 45
2.6. Frente de Pareto y población del ejemplo 2.1. 49
2.7. Calculo del *crowding* para cada individuo. 53
2.8. Calculo del *crowding* de cada individuo por medio de polígonos. 55
2.9. Diagrama del operador de selección de supervivientes del NSGA-II. . . . 59
2.10. Comparativo de dos aproximaciones distintas de un frente de Pareto. . . 64
2.11. Aproximación de $\mathcal{FP}^*$ del ZDT3 usando el NSGA-II. 68
2.12. Indicadores de desempeño para el MOP ZDT3. 69
2.13. Comportamiento de Δ_p en las últimas 20 generaciones para la ZDT3. . . 70

3.1. Regla de Armijo para distintos valores de c. 74
3.2. Ejemplo gráfico del MDPM usando la función de Booth. 75
3.3. Comparativa de frentes e indicador de desempeño, NSGA-II Original vs. Híbrido para MOP ZDT1. 82
3.4. Comparativa de frentes e indicador de desempeño, NSGA-II Original vs. Híbrido para MOP ZDT2. 83
3.5. Comparativa de frentes e indicador de desempeño, NSGA-II Original vs. Híbrido para MOP ZDT3. 84
3.6. Comparativa de frentes e indicador de desempeño, NSGA-II Original vs. Híbrido para MOP ZDT4. 85
3.7. Comparativa de frentes e indicador de desempeño, NSGA-II Original vs. Híbrido para MOP ZDT6 86
3.8. Comparativo NSGA-II vs. Híbrido para MOPs de prueba con distintos ϵ. 88

A.1. Frentes de Pareto correspondientes a los MOPs ZDT1, ZDT2, ZDT3, ZDT4 y ZDT6. 97

Índice de tablas

1.1. Proceso evolutivo VS problema. 7
1.2. Parámetros usados en un AG. 10
1.3. Parámetros usados en el ejemplo 1.2. 10
1.4. Población inicial del ejemplo 1.2. 12
1.5. Primera población del ejemplo 1.2 21
1.6. Parámetros usados en la función de Beale. 27
1.7. Parámetros usados en la función de Rastrigin. 29
1.8. Parámetros usados en la función de Rosenbrock. 31
1.9. Parámetros usados en la función de Rosenbrock para $n = 10$. 32

2.1. Costo y rendimiento de los equipos. 39
2.2. Población del ejemplo 2.1. 50
2.3. S_p y n_p correspondiente a cada individuo del ejemplo 2.1. 51
2.4. Comparativo entre indicadores de desempeño. 67
2.5. Parámetros usados en el NSGA-II para aproximar $\mathcal{FP}^*$ del ZDT3 68

3.1. Parámetros de las funciones de prueba usadas en la sección 3.2.1. 81
3.2. Resultados experimentales NSGA-II Original vs. Híbrido. 82
3.3. Comparativa de rendimiento entre los algoritmos NSGA-II e Híbrido. . . 87

Índice de algoritmos

1. Algoritmo para aproximación de rutas optimas (ascenso por pasos). 3
2. Algoritmo genético simple. 6
3. Algoritmo de selección por torneo para un AG mono-objetivo. 13

4. Clasificación veloz de elementos no dominados. 48
5. Asignación de medida de densidad. 53
6. Operador de selección y cruza del NSGA-II. 57
7. Operador de selección de supervivientes para el NSGA-II. 59
8. NSGA-II. 61

9. Método de descenso con pendiente máxima. 78
10. NSGA-II Híbrido. 79

Lista de Acrónimos

AG *Algoritmo Genético*

IA Inteligencia Artificial

AE *Algoritmos Evolutivo*

PE Programación Evolutiva

EE Estrategia Evolutiva

MOP Problema de Optimización Multi-Objetivo

VEGA Vector Evaluated Genetic Algorithm

MOEA Algoritmo Evolutivo Multi-Objetivo

NSGA Non-dominated Sorting Genetic Algorithm

NSGA-II Non-dominated Sorting Genetic Algorithm II

SPEA-II Strength Pareto Evolutionary Algorithm II

SMS-EMOA Multiobjective Selection Based on Dominated Hypervolume

CTSO Crowded Tournament Selection Operator

OSS Operador de Selección de Supervivientes

OSC Operador de Selección y Cruza

MDPM Método de Descenso con Pendiente Máxima

Bibliografía

[AMYA12] Ehsan Arianyan, Davood Maleki, Alireza Yari, and Iman Arianyan. Efficient resource allocation in cloud data centers through genetic algorithm. In *Telecommunications (IST), 2012 Sixth International Symposium on*, pages 566–570. IEEE, 2012.

[BA03] Barrie M Baker and MA Ayechew. A genetic algorithm for the vehicle routing problem. *Computers & Operations Research*, 30(5):787–800, 2003.

[Bar62] Nils Aall Barricelli. Numerical testing of evolution theories. *Acta Biotheoretica*, 16(1):69–98, 1962.

[BNE07] Nicola Beume, Boris Naujoks, and Michael Emmerich. Sms-emoa: Multiobjective selection based on dominated hypervolume. *European Journal of Operational Research*, 181(3):1653–1669, 2007.

[Che12] Shu-Heng Chen. *Genetic algorithms and genetic programming in computational finance*. Springer Science & Business Media, 2012.

[CLV06] Carlos A. Coello Coello, Gary B. Lamont, and David A. Van Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems (Genetic and Evolutionary Computation)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[Coe99] Carlos A. Coello Coello. La importancia de la representación en los algoritmos genéticos (parte 1). *Soluciones Avanzadas. Tecnologías de Información y Estrategias de Negocios*, VII(69):50–56, 1999.

[CRLL12] Timur Chabuk, James Reggia, Jason Lohn, and Derek Linden. Causally-guided evolutionary optimization and its application to antenna array design. *Integrated Computer-Aided Engineering*, 19(2):111–124, 2012.

[DA94] Kalyanmoy Deb and Ram B Agrawal. Simulated binary crossover for continuous search space. *Complex Systems*, 9(3):1–15, 1994.

[DG96] Kalyanmoy Deb and Mayank Goyal. A combined genetic adaptive search (geneas) for engineering design. *Computer Science and Informatics*, 26:30–45, 1996.

[DgB95] Kalyanmoy Deb and Hans georg Beyer. Self-adaptive genetic algorithms with simulated binary crossover. *Complex Systems*, 9:431–454, 1995.

[DK01] Kalyanmoy Deb and Deb Kalyanmoy. *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons, Inc., New York, NY, USA, 2001.

[DPAM02] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *Evolutionary Computation, IEEE Transactions on*, 6(2):182–197, 2002.

[ES03] Agoston E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. SpringerVerlag, 2003.

[Fra60] Alex S Fraser. Simulation of genetic systems by automatic digital computers vi. epistasis. *Australian Journal of Biological Sciences*, 13(2):150–162, 1960.

[GH88] David E Goldberg and John H Holland. Genetic algorithms and machine learning. *Machine learning*, 3(2):95–99, 1988.

[GJ06] Robin R Gutell and Robert K Jansen. Genetic algorithm approaches for the phylogenetic analysis of large biological sequence datasets under the maximum likelihood criterion. 2006.

[Gol89] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989.

[HIK+96] Tetsuya Higuchi, Masaya Iwata, Isamu Kajitani, Hitoshi Yamada, Bernard Manderick, Yuji Hirao, Masahiro Murakawa, Shuji Yoshizawa, and Tatsumi Furuya. Evolvable hardware with genetic learning. In *Circuits and Systems, 1996. ISCAS'96., Connecting the World., 1996 IEEE International Symposium on*, volume 4, pages 29–32. IEEE, 1996.

[Hol75] John H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, 1975. second edition, 1992.

[LCS10] Adriana Lara López, Carlos A Coello Coello, and Oliver Schuetze. A painless gradient-assisted multi-objective memetic mechanism for solving continuous bi-objective optimization problems. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8. IEEE, 2010.

[LLY+08] Xingtao Liao, Qing Li, Xujing Yang, Weigang Zhang, and Wei Li. Multiobjective optimization for crash safety design of vehicles using stepwise regression model. *Structural and Multidisciplinary Optimization*, 35(6):561–569, 2008.

[LY08] D.G. Luenberger and Y. Ye. *Linear and Nonlinear Programming*. International Series in Operations Research & Management Science. Springer US, 2008.

[MSRM93] Eric Michielssen, J-M Sajer, S Ranjithan, and Raj Mittra. Design of lightweight, broad-band microwave absorbers using genetic algorithms. *IEEE Transactions on Microwave Theory and Techniques*, 41(6):1024–1031, 1993.

[Mun99] James Munkres. *Topology*. Prentice Hall, 1999.

[NW06] Jorge Nocedal and Stephen Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer, New York, NY, second edition, 2006.

[PGH71] Charles J. Stone Paul G. Hoel, Sidney C. Port. *Introduction to Probability Theory*. Cengage Learning, 1 edition, 1971.

[SD94] Nidamarthi Srinivas and Kalyanmoy Deb. Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary computation*, 2(3):221–248, 1994.

[SELC12] Oliver Schütze, Xavier Esquivel, Adriana Lara, and Carlos A Coello Coello. Using the averaged hausdorff distance as a performance measure in evolutionary multiobjective optimization. *Evolutionary Computation, IEEE Transactions on*, 16(4):504–522, 2012.

[SFH+03] Kisung Seo, Zhun Fan, Jianjun Hu, Erik D Goodman, and Ronald C Rosenberg. Toward a unified and automated design methodology for multi-domain dynamic systems using bond graphs and genetic programming. *Mechatronics*, 13(8):851–885, 2003.

[Tur50] Alan M. Turing. Computing machinery and intelligence. *MIND A QUARTERLY REVIEW OF PSYCHOLOGY AND PHILOSOPHY*, LIX(236):433–460, 1950.

[ZLT01] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. Spea2: Improving the strength pareto evolutionary algorithm. Technical report, 2001.

[ZM04] David B. Fogel Zbigniew Michalewicz. *How to Solve It: Modern Heuristics*. Springer-Verlag, Berlin, 2 edition, 2004.