UNIVERSIDAD CENTROCCIDENTAL
"LISANDRO ALVARADO"

Decanato de Ciencias y Tecnología
Licenciatura en Ciencias Matemáticas

"UN ALGORITMO EVOLUTIVO PARA PROGRAMACIÓN LINEAL MULTIOBJETIVO"

TRABAJO ESPECIAL DE GRADO PRESENTADO POR

BR. ISAAC J. MENDOZA

COMO REQUISITO FINAL
PARA OBTENER EL TÍTULO DE LICENCIADO
EN CIENCIAS MATEMÁTICAS
ÁREA DE CONOCIMIENTO: **OPTIMIZACIÓN MULTIOBJETIVO**
TUTOR: DR. RÓMULO CASTILLO

Barquisimeto, Venezuela. Noviembre de 2008

Universidad Centroccidental
"Lisandro Alvarado"
Decanato de Ciencias y Tecnología
Licenciatura en Ciencias Matemáticas

ACTA
TRABAJO ESPECIAL DE GRADO

Los suscritos miembros del Jurado designados por el Jefe del Departamento de Matemáticas del Decanato de Ciencias y Tecnología de la Universidad Centroccidental "Lisandro Alvarado", para examinar y dictar el veredicto sobre el Trabajo Especial de Grado titulado:
"UN ALGORITMO EVOLUTIVO PARA PROGRAMACIÓN LINEAL MULTIOBJETIVO"

Presentado por el ciudadano BR. ISAAC J. MENDOZA titular de la Cédula de Identidad Nº V-16.385.532. Con el propósito de cumplir con el requisito académico final para el otorgamiento del título de Licenciado en Ciencias Matemáticas.

Luego de realizada la Defensa y en los términos que imponen los Lineamientos para el Trabajo Especial de Grado de la Licenciatura en Ciencias Matemáticas, se procedió a discutirlo con el interesado habiéndose emitido el veredicto que a continuación se expresa:

[1] _____

Con una calificación de _____ puntos.

En fe de lo expuesto firmamos la presente Acta en la Ciudad de Barquisimeto a los _____ días del mes de _____ de _____.

_____	_____
TUTOR	FIRMA
_____	_____
JURADO	FIRMA
_____	_____
JURADO	FIRMA

OBSERVACIONES:

_____

_____

_____

[1] Aprobado ó Reprobado

Universidad Centroccidental "Lisandro Alvarado"

"Un Algoritmo Evolutivo para Programación Lineal Multiobjetivo"

Br. Isaac J. Mendoza

Tutor: Dr. Rómulo Castillo

Resumen

Los problemas del mundo real, en su gran mayoría presentan diferentes objetivos a optimizarse (los cuales generalmente se expresan en unidades diferentes y están en conflicto entre sí) y un espacio de búsqueda grande y complejo (p.ej., no convexo, no diferenciable, etc.). Estas dos características son suficientes para que los métodos tradicionales de optimización resulten inoperantes o simplemente requieran un costo computacional prohibitivo.

En este trabajo se aborda el problema de programación lineal multiobjetivo, se establecen las condiciones necesarias para caracterizar el conjunto eficiente de soluciones (condiciones de Pareto) y se realiza el estudio teórico que permite analizar los diferentes enfoques tradicionales para encontrar soluciones. Aunque por lo expuesto al principio de este resumen, este tipo de problemas requieren de técnicas alternativas de solución.

Por otra parte, se realiza un estudio general de los algoritmos evolutivos, las bases que los sustentan y su adaptación al problema considerado. Se crea un código computacional basado en algoritmos evolutivos para evaluar el desempeño en algunos problemas prácticos considerando a su vez algunas variaciones o ajustes que permitan un mejor desempeño.

Palabras Clave: Multiobjetivo, algoritmo, evolutivo, programación lineal, ruta corta.

A Dios Todopoderoso. Yahweh Meshia.

If we knew what we were doing, it wouldn't be called research.

– Albert Einstein

AGRADECIMIENTOS

Resulta difcil recordar en este momento a todas las personas que han colaborado en mayor o menor medida a que este trabajo llegue a su fin, pues han sido muchas.

Agradezco primeramente a Mi Único y Gran Rey, el Señor de Señores Jesús de Nazareth, sin ti no podría haber realizado este trabajo.

A tí Victorina, por haberme traído al mundo, y por tus Grandes esfuerzos durante todos estos años. Te Amo!!!.

A mis hermanos: Neila, Abigail y Evlin, por apoyarme en todo momento de mi carrera. A todos mis familiares que por razones de espacio no cabrían acá, a todos ellos Gracias, mil Gracias.

A mi ***Pincha***, quien me ha acompañado, ayudado, comprendido, etc. etc. prácticamente durante toda mi carrera, has sido un pilar importante no solo en mis estudios sino también en mi vida personal y espiritual con Dios. No podría dejar de decirte que Te Amo!!!, a Rosa Leal, que verdaderamente le hace honor tanto a su nombre como a su apellido, una vez más Te Amo!!!.

A mis amigos del alma, con quienes compartí muchos momentos tanto de tristeza como también esos momentos en que casi de tanto reirnos teníamos que ir al baño, aunque no pueda recordar todos sus nombres pues le pido disculpas, entre ellos están, Bartolo, Yolima, Luisa, Maira, Maira Castro, Isa, María Carruido, María Rodríguez, Andy El Achouche y pues los otros quedarán para la Tesis de Postgrado, Dios mediante.

A mi ***asesor***, porque tutor se queda pequeño, y amigo, que desde que concursé para Peparador del Laboratorio de Matemáticas me ha ayudado y siempre me ha brindado su apoyo, sin él este trabajo no tendría esos detalles que solo con su experiencia están incluídos acá, con toda mi humildad muchas gracias por todo, al Dr. Rómulo Castillo, verdaderamente un gran hombre.

A todos aquellos profesores que de una u otra manera han colaborado y contribuído de verdad a mi desarrollo académico y personal, en particular quiero agradecer muy especialmente al Profesor Pedro Adames, quien junto con mi asesor Rómulo, han sido los profesores que más me han ayudado, en verdad lamento su partida mi gran amigo.

ÍNDICE

Agradecimientos **i**

Introducción **1**

Capítulo 1. Optimización Con Múltiples Objetivos **3**

1.1. Introducción . . . 3
1.2. Conceptos básicos y terminología . . . 4
1.3. Búsqueda y Toma de Decisiones . . . 8
1.4. Métodos tradicionales para resolver MOPs . . . 10
1.4.1. Método de la Suma Ponderada . . . 11
1.4.2. Método de la ε-restricción . . . 13
1.4.3. Discusión de los métodos clásicos . . . 14
1.5. Algoritmos evolutivos en Optimización Multiobjetivo . . . 15
1.5.1. Algoritmos Genéticos . . . 17
1.5.2. Algoritmos Evolutivos y optimización multiobjetivo . . . 18
1.6. Conclusiones finales y delineamiento de capítulos posteriores . . . 20

Capítulo 2. Algoritmos Evolutivos **21**

2.1. Introducción . . . 21
2.2. El Algoritmo Genético Simple (AGS) . . . 23
2.2.1. Codificación . . . 23
2.2.2. Ejemplo . . . 26
2.3. Extensiones y Modificaciones del Algoritmo Genético Simple . . . 27
2.4. Población . . . 28
2.4.1. Tamaño de la población . . . 28
2.4.2. Población inicial . . . 28
2.5. Función objetivo . . . 29
2.6. Selección . . . 30
2.7. Cruce . . . 31
2.8. Mutación . . . 31
2.9. ¿Por qué funcionan? . . . 32
2.9.1. Teorema de los esquemas . . . 32
2.9.2. Paralelismo Implícito . . . 35

2.9.3. Teoremas de Convergencia 35

Capítulo 3. Algoritmos Evolutivos para Optimización Múltiobjetivo 37
3.1. Definición del Problema de Red 37
3.2. El Problema de la Ruta Corta 38
3.3. Condiciones Estocásticas 39
3.4. Algoritmos Evolutivos Estocásticos 39
3.5. Redes Estocásticas Multiobjetivo 39
3.6. Simulación de Variables Aleatorias 40
3.6.1. Distribuciones Continuas 40
3.6.2. Distribuciones Discretas 41
3.7. Programa 42
3.7.1. Algoritmo que Resuelve el Problema de un Objetivo Determinístico 42
3.7.2. Algoritmo que Resuelve el Problema de un Objetivo Estocástico 43
3.7.3. Algoritmo que Resuelve el Problema Multiobjetivo 43
3.7.4. Algoritmo que Resuelve el Problema Multiobjetivo de Ruta Corta Estocástico 44
3.8. Datos 44
3.9. Algoritmo 45
3.10. Estructuras que Ayudan a los Algoritmos Genéticos 46
3.10.1. Rutas Aleatorias y Estructuras Dinámicas 46
3.10.2. Estructura Dinámica de Cromosomas 47
3.10.3. Lista Adyacente 47
3.11. Búsqueda de Soluciones en un Espacio Estocástico Utilizando EA 48
3.11.1. Población Inicial 48
3.11.2. Encontrando un Frente de Pareto 49
3.12. Resultados 49

Capítulo 4. Resultados 51
4.1. Introducción 51
4.2. Resultados Determinísticos 51
4.3. Resultados de la Red Estocástica 53

Conclusiones 57

Referencias 59

Apéndice A. Archivo de Entrada 63

Apéndice B. Código Rutas Cortas 65

Apéndice C. Manual del Programa **83**
C.1. Programa paths3.cpp 83
C.2. Objetivos 83
C.3. Restricciones de las Redes 84
C.4. Cambio en los Parámetros 84
C.5. Formato del Archivo de Entrada 84

ÍNDICE DE TABLAS

2.2.1. Población inicial de la simulación efectuada a mano correspondiente al Algoritmo Genético Simple 25
2.2.2. Población en el tiempo 1 26

3.8.1. Claves asignadas a las Distribuciones 46
3.8.2. Representación de la Entrada 46

4.2.1. Población inicial para el problema determinístico 52
4.2.2. Frente de Pareto del caso determinístico 53
4.3.1. Conjunto de soluciones caso estocástico (100G, 50 % recombinación, 50 % mutación) (Corrida I) 54
4.3.2. Conjunto de soluciones caso estocástico (100G, 50 % recombinación, 50 % mutación) (Corrida II) 55
4.3.3. Medias y desviaciones estándar obtenidas a partir de los 10 frentes de Pareto 56

ÍNDICE DE FIGURAS

1.2.1. Ejemplo de la optimalidad Pareto en el espacio objetivo. 5
1.2.2. Valores de f_1 y f_2 en función de $x \in \boldsymbol{X}_f = (-\infty, \infty)$ 8
1.2.3. Frente Pareto del MOP 9

2.2.1. Operador de cruce basado en un punto 24
2.2.2. Operador de mutación 25

3.7.1. Diagrama de flujo para encontrar soluciones de Pareto 50

4.2.1. Red determinística con 16 nodos 52
4.3.1. Frente de Pareto para una red estocástica de 16 nodos y 27 arcos utilizando distribuciones Normales entre los arcos (Corrida I) 54
4.3.2. Frente de Pareto para una red estocástica de 16 nodos y 27 arcos utilizando distribuciones Normales entre los arcos (Corrida II) 55
4.3.3. Frentes de Pareto con funciones normales (0, 1) en los arcos. 56

ÍNDICE DE PSEUDOCÓDIGOS

2.2.1. Pseudocódigo del Algoritmo Genético Básico 23
2.3.1. Pseudocódigo del Algoritmo Genético Abstracto 28
3.9.1. Pseudocódigo del Algoritmo Evolutivo para la Ruta más Corta Estocástica Multiobjetivo 47

INTRODUCCIÓN

Como una parte muy importante de las Matemáticas Aplicadas, se encuentra el desarrollo e implementación de modelos para la solución de problemas de optimización en la empresa e industria. La optimización puede contemplar objetivos aislados o múltiples. El problema que se resolvió tiene objetivos múltiples. Los diferentes trabajos de investigación sobre optimización multiobjetivo, regularmente se han resuelto tratándolos como problemas de optimización con un único objetivo utilizando métodos que ya son clásicos o tradicionales en esta rama, transformando todos los objetivos, excepto uno, en restricciones, estos métodos son abordados en este trabajo, y también los conceptos relacionados con la definición de «***óptimo***» para problemas multiobjetivo. Cabe destacar que también se har realizado trabajos en los que se han resuelto problemas donde las restricciones están dadas por valores determinísticos (NISHIMURA y MATSUMOTO, 1998).

En este trabajo se resolvió un problema de ruta corta estocástico multiobjetivo. Se trata de un problema importante ya que sabiendo resolver este, se pueden resolver algunos otros problemas, tales como la planeación de proyectos. Lo anterior se logra al adecuar los datos de este tipo de problemas a una red de ruta corta. En el problema de ruta corta se cuenta con una red de varios nodos que pueden representar «*ciudades*» y varios arcos o caminos que unen a dichas ciudades con varios costos asociados, que en este caso son variables. Al contar con muchos costos en un arco, se convierte en un problema multiobjetivo. Lo que se busca es encontrar las mejores rutas para llegar de un punto a otro minimizando los costos.

Para resolver el problema se creó un algoritmo aplicando optimización multiobjetivo con costos aleatorios al algoritmo previamente desarrollado por el Dr. GÓMEZ-SÁNCHEZ en 2001, que resolvió el problema estocástico con un único objetivo. Se obtuvieron vectores de soluciones, cuyos valores son aceptables para todas las funciones objetivo, mostrándolos a través de diagramas de Pareto que representan dichas soluciones.

Como objetivo general se tuvo la creación de un algoritmo implementado en un lenguaje de programación (C++) que resuelve el problema. Las diferentes pruebas obtenidas del programa dieron origen a varios diagramas de Pareto, donde se puede analizar el comportamiento de los diferentes objetivos y parámetros.

Para lograr lo anterior se consideraron los siguientes objetivos específicos:

1. Crear rutas aleatorias para el problema de ruta corta multiobjetivo.
2. Evaluar dichas rutas para conocer los valores de las funciones objetivo.

3. Generar diagramas de Pareto.

4. Evaluar diferentes combinaciones de parámetros en el algoritmo genético.

Los alcances del proyecto consideraron la creación de programas para construir diagramas de Pareto, donde se mostraron las soluciones para el problema de ruta corta multiobjetivo, evaluando las diferentes combinaciones de parámetros en el algoritmo evolutivo.

Las delimitaciones son: se trata de un caso particular de optimización (ruta corta multiobjetivo) y se utilizaron variables aleatorias con distribuciones conocidas, típicas dentro de la investigación. Las limitaciones se encontraron en el tamaño de redes que se estudiaron (sujeto a la revisión de literatura que se hizo), así como dentro del lenguaje de programación que se utilizó. Como punto final se presentaron diagramas de Pareto para su interpretación.

Para lograr los objetivos planteados al comenzar este proyecto, se revisó la literatura existente referente al problema, para ver los intentos actuales que se han estado realizando para resolver este problema. Posteriormente se construyó un algoritmo que resuelve el problema de ruta corta multiobjetivo, usando estructuras variables de datos, utilizando la programación en el lenguaje C++ donde se implementó el algoritmo para la solución del problema.

El trabajo está organizado en cuatro capítulos. En el capítulo 1 se describen todos los conceptos necesarios sobre la teoría de optimización con múltiples objetivos, dando una introducción a la optimización, definiciones básicas para crear los diagramas de Pareto, y discutiendo los métodos más usados para resolver este tipo de problemas. En el capítulo 2 se explica con sumo detalle la técnica de los algoritmos genéticos para resolver problemas de optimización. En el capítulo 3 se definen los conceptos básicos del problema lineal multiobjetivo planteado y se describe la forma como se utilizaron los conceptos y técnicas para programar y resolver el problema. En el capítulo 4 se desarrollan los resultados obtenidos de las pruebas que se realizaron con el programa, así como las comparaciones entre los resultados. En el último capítulo se exponen las conclusiones obtenidas de la investigación. Como anexo se integra el código del programa desarrollado, además se entrega un Programa Ejecutable y un archivo de entrada del problema planteado.

CAPÍTULO 1

OPTIMIZACIÓN CON MÚLTIPLES OBJETIVOS

§1.1. INTRODUCCIÓN

En la vida siempre se enfrentan problemas en donde se deben tomar decisiones, tratando de llegar a un objetivo. La optimización es una herramienta que facilita dicha tarea. Sin embargo, muchos de los problemas de la vida real envuelven optimización simultánea de diferentes objetivos en conflicto. Este tipo de problemas es llamado optimización multiobjetivo o vectorial y fue estudiado originalmente en el área económica. También los científicos e ingenieros se han dado cuenta de que este tipo de problemas se encuentra en todas las áreas de estudio. Sin embargo, al tratarlos como problemas con un único objetivo, como se resuelven regularmente por la facilidad, en la toma de decisiones, no se puede observar claramente el conjunto de alternativas óptimas cuando se busca la solución.

En el transcurso de los años, se han estado desarrollando técnicas para manejar los problemas de optimización multiobjetivo. Pero no ha sido hasta hace unos 20 años en que investigadores han utilizado el potencial de los Algoritmos Evolutivos (Evolutionary Algorithms, EA) y otras heurísticas basadas en poblaciones.

Por otro lado, los problemas se han resuelto de forma determinística, lo cual, no es común en la vida real, ya que gran parte de las situaciones dependen de factores aleatorios, lo que hace que los costos, por ejemplo, cambien como si fueran variables estocásticas.

Muchas aplicaciones para la toma de decisiones tales como el ruteo de vehículos, alternativas de localización, programación y evaluación de proyectos implican condiciones estocásticas debido a los cambios en el ambiente. Además, es muy común encontrar funciones objetivo múltiples para evaluar cuando nos estamos ocupando de problemas más realistas. Muchas técnicas evolutivas se han aplicado con éxito a los problemas bajo condiciones determinísticas y probabilísticas (AHUJA y otros, 1993; AVERBAKH y BERMAN, 1998; BAZARAA y otros, 1990; CHEN y otros, 2001; DAVIS y LINGRAS, 2003; GOLDBERG, 1989).El uso de una representación particular en un algoritmo genético es importante para el funcionamiento del algoritmo (GOLDBERG, 1989). La segunda generación de algoritmos genéticos multiobjetivo que se detallan en la sección 1.5.2 han estado incorporando procedimientos para filtrar y reducir el número de individuos a evaluar y permitir un mejor funcionamiento en la búsqueda de los frentes de Pareto.

§1.2. CONCEPTOS BÁSICOS Y TERMINOLOGÍA

DEFINICIÓN 1.2.1 (Optimización)**.** La optimización es una rama de las matemáticas aplicadas que consiste en la creación y elaboración de principios y métodos usados para solucionar problemas cuantitativos de muchas disciplinas como física, biología, ingeniería y economía para obtener la mejor o una buena solución.

Existen diferentes problemas de optimización, que los podemos dividir en dos grandes grupos de acuerdo al número de objetivos que se tratan de resolver: los problemas de optimización simple y los problemas de optimización multiobjetivo. La diferencia más grande que se podría encontrar entre los anteriores es que los de optimización simple buscan obtener el mejor diseño o decisión, el cual es regularmente un máximo o mínimo global, según sea el caso de maximizar o minimizar. En cambio en la optimización multiobjetivo puede no existir una solución que sea la mejor con respecto a todos los objetivos. Existe un conjunto de soluciones que son las mejores del resto cuando todos los objetivos son considerados pero no tan buenas a otras soluciones al tomar en cuenta uno o más objetivos.

DEFINICIÓN 1.2.2 (Problema de Optimización Multiobjetivo)**.** Un Problema de Optimización Multiobjetivo (MOP, por sus siglas en Inglés) general incluye un conjunto de n parámetros (variables de decisión), un conjunto de k funciones objetivo, y un conjunto de m restricciones. Las funciones objetivo y las restricciones son funciones de las variables de decisión. Luego, el MOP puede expresarse como:

$$\begin{aligned} \text{Optimizar} \quad & \boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_k(\boldsymbol{x})) \\ \text{sujeto a} \quad & \boldsymbol{e}(\boldsymbol{x}) = (e_1(\boldsymbol{x}), e_2(\boldsymbol{x}), \ldots, e_m(\boldsymbol{x})) \\ \text{donde} \quad & \boldsymbol{x} = (x_1, x_2, \ldots, x_n) \in \boldsymbol{X} \\ & \boldsymbol{y} = (y_1, y_2, \ldots, y_n) \in \boldsymbol{Y} \end{aligned} \tag{1.2.1}$$

siendo $\boldsymbol{x}$ el vector de decisión e $\boldsymbol{y}$ el vector objetivo. El espacio de decisión se denota por $\boldsymbol{X}$, y al espacio objetivo por $\boldsymbol{Y}$. Optimizar, dependiendo del problema, puede significar igualmente, minimizar o maximizar.

El conjunto de restricciones $\boldsymbol{e}(\boldsymbol{x}) \geqslant 0$ determina el conjunto de soluciones factibles $\boldsymbol{X}_f$ y su correspondiente conjunto de vectores objetivo factibles $\boldsymbol{Y}_f$.

DEFINICIÓN 1.2.3 (Conjunto de Soluciones Factibles)**.** El conjunto de soluciones factibles $\boldsymbol{X}_f$ se define como el conjunto de vectores de decisión $\boldsymbol{x}$ que satisface los requerimientos $\boldsymbol{e}(\boldsymbol{x})$:

$$\boldsymbol{X}_f = \{\boldsymbol{x} \in \boldsymbol{X} | \boldsymbol{e}(\boldsymbol{x}) \geqslant 0\} \tag{1.2.2}$$

La imagen de $\boldsymbol{X}_f$, es decir, la región factible del espacio objetivo, se denota por

$$\boldsymbol{Y}_f = \boldsymbol{f}(\boldsymbol{X}_f) = \bigcup_{x \in X_f} \{\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})\} \tag{1.2.3}$$

De estas definiciones se tiene que cada solución del *MOP* en cuestión consiste de una n-upla $\boldsymbol{x} = (x_1, x_2, ..., x_n)$, que conduce a un vector objetivo $\boldsymbol{y} = (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), ..., f_k(\boldsymbol{x}))$, donde cada $\boldsymbol{x}$ debe cumplir con el conjunto de restricciones $\boldsymbol{e}(\boldsymbol{x}) \geqslant 0$. El problema de optimización consiste en hallar la $\boldsymbol{x}$ que tenga el «***mejor valor***» de $\boldsymbol{f}(\boldsymbol{x})$. En general, y según ya se ha introducido, no existe un único «***mejor valor***», sino un conjunto de soluciones. Entre éstas, ninguna se puede considerar mejor a las demás si se tienen en cuenta todos los objetivos al mismo tiempo. Este hecho deriva de que puede existir –y generalmente existe– conflicto entre los diferentes objetivos que componen el problema. Por ende, al tratar con *MOPs* se precisa de un nuevo concepto de «***óptimo***».

En la optimización de un solo objetivo el conjunto de variables de decisión factibles está completamente ordenado mediante una función objetivo f. Es decir, dadas dos soluciones $\boldsymbol{a}, \boldsymbol{b} \in \boldsymbol{X}_f$, se cumple una sola de las siguientes proposiciones: $f(\boldsymbol{a}) > f(\boldsymbol{b}), f(\boldsymbol{a}) = f(\boldsymbol{b})$ ó $f(\boldsymbol{b}) > f(\boldsymbol{a})$. El objetivo consiste en hallar la solución (o soluciones) que tengan los valores óptimos (máximos o mínimos) de f. Cuando se trata de varios objetivos, sin embargo, la situación cambia. $\boldsymbol{X}_f$, en general, no está totalmente ordenada por los objetivos; el orden que se da suele ser parcial (i.e. existen vectores de decisión $\boldsymbol{a}$ y $\boldsymbol{b}$ con los que $\boldsymbol{f}(\boldsymbol{a})$ no puede considerarse mejor que $\boldsymbol{f}(\boldsymbol{b})$ y tampoco $\boldsymbol{f}(\boldsymbol{b})$ puede considerarse mejor que $\boldsymbol{f}(\boldsymbol{a})$).

EJEMPLO 1.2.1 (Problema con dos objetivos en Conflicto)**.** El siguiente gráfico muestra la relación entre dos funciones f_1 y f_2. La función f_1 representa el costo de los dispositivos de seguridad de un sistema, mientras que f_2 representa el índice de ocurrencia de accidentes.

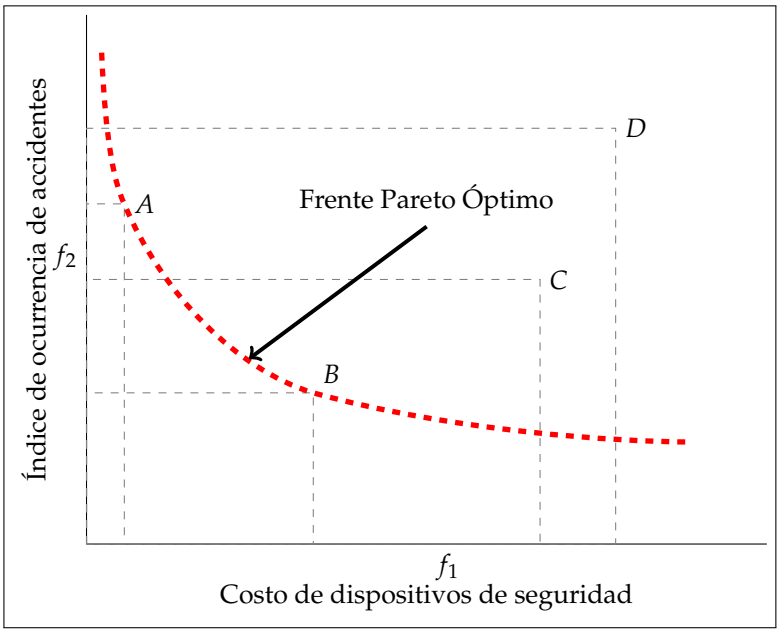


FIGURA 1.2.1: Ejemplo de la optimalidad Pareto en el espacio objetivo.

La solución representada por el punto B es mejor que la representada por el punto C, debido a que provee de una mayor seguridad a un costo menor. La solución B sería también la escogida en

el caso de estar realizando la optimización de un solo objetivo. Más aún, todas las soluciones que se encuentran en el rectángulo delimitado por el origen de coordenadas y la solución C y por encima de la curva, son mejores a C. En la comparación entre C y A, se obtiene que, disminuyendo mucho los costos, el nivel de seguridad provisto por A es solo ligeramente peor, aunque C y A son no comparables entre ellos porque no se podría argumentar que uno es mejor que otro al considerar todos los objetivos. Si comparamos a A con B tampoco podríamos establecer que alguna de las dos sea mejor, si se considera que ambos objetivos son igualmente importantes y no se introduce alguna consideración de índole subjetiva. Sin embargo, B es claramente superior a C en ambos objetivos.

Para poder expresar estas comparaciones matemáticamente, las relaciones $=$, $\leqslant$ y $\geqslant$ se deben extender.

Esto se puede realizar de la siguiente manera:

DEFINICIÓN 1.2.4. Dados dos vectores de decisión $\boldsymbol{u} \in \boldsymbol{X}$ y $\boldsymbol{v} \in \boldsymbol{X}$,

$$\begin{aligned} \boldsymbol{f}(\boldsymbol{u}) &= \boldsymbol{f}(\boldsymbol{v}) \text{ si y solo si } \forall i \in \{1,2,\ldots,k\} : f_i(\boldsymbol{u}) = f_i(\boldsymbol{v}) \\ \boldsymbol{f}(\boldsymbol{u}) &\geqslant \boldsymbol{f}(\boldsymbol{v}) \text{ si y solo si } \forall i \in \{1,2,\ldots,k\} : f_i(\boldsymbol{u}) \geqslant f_i(\boldsymbol{v}) \\ \boldsymbol{f}(\boldsymbol{u}) &> \boldsymbol{f}(\boldsymbol{v}) \text{ si y solo si } \forall i \in \{1,2,\ldots,k\} : f_i(\boldsymbol{u}) > f_i(\boldsymbol{v}) \end{aligned} \tag{1.2.4}$$

Las relaciones $\leqslant$ y $<$ se definen de manera similar.

A partir de esta noción, se sigue que $f(B) < f(C)$, $f(C) < f(D)$, y como consecuencia $f(B) < f(D)$. Sin embargo, al comparar A y C o A y B, no se puede decir que alguna sea superior a la otra. Por ejemplo, a pesar de que la solución representada por B es más cara, provee menor índice de accidentes que la representada por A.

Por lo expuesto, se tiene que dos vectores de decisión x_1 y x_2 de un *MOP* pueden cumplir solo una de tres condiciones posibles: $f(x1) < f(x2)$, $f(x2) < f(x1)$ o $f(x1) \nleqslant f(x2) \wedge f(x2) \nleqslant f(x1)$. Esta situación se expresa con los siguientes símbolos y términos:

DEFINICIÓN 1.2.5 (Dominancia Pareto en un contexto de Maximización)**.** Para dos vectores objetivo $\boldsymbol{a}$ y $\boldsymbol{b}$,

$$\begin{aligned} \boldsymbol{a} \succ \boldsymbol{b} (\boldsymbol{a} \text{ domina a } \boldsymbol{b}) \qquad & \text{si y sólo si } \boldsymbol{a} > \boldsymbol{b} \\ \boldsymbol{b} \succ \boldsymbol{a} (\boldsymbol{b} \text{ domina a } \boldsymbol{a}) \qquad & \text{si y sólo si } \boldsymbol{b} > \boldsymbol{a} \\ \boldsymbol{a} \sim \boldsymbol{b} (\boldsymbol{a} \; y \; \boldsymbol{b} \text{ no son comparables}) \qquad & \text{si y sólo si } \boldsymbol{a} \ngeqslant \boldsymbol{b} \wedge \boldsymbol{b} \ngeqslant \boldsymbol{a} \end{aligned} \tag{1.2.5}$$

DEFINICIÓN 1.2.6 (Dominancia Pareto en un contexto de Minimización)**.** Para dos vectores objetivo $\boldsymbol{a}$ y $\boldsymbol{b}$,

$$\begin{aligned} \boldsymbol{a} \succ \boldsymbol{b} (\boldsymbol{a} \text{ domina a } \boldsymbol{b}) \qquad & \text{si y sólo si } \boldsymbol{a} < \boldsymbol{b} \\ \boldsymbol{b} \succ \boldsymbol{a} (\boldsymbol{b} \text{ domina a } \boldsymbol{a}) \qquad & \text{si y sólo si } \boldsymbol{b} < \boldsymbol{a} \\ \boldsymbol{a} \sim \boldsymbol{b} (\boldsymbol{a} \; y \; \boldsymbol{b} \text{ no son comparables}) \qquad & \text{si y sólo si } \boldsymbol{a} \nleqslant \boldsymbol{b} \wedge \boldsymbol{b} \nleqslant \boldsymbol{a} \end{aligned} \tag{1.2.6}$$

Luego, de aquí en adelante, ya no será necesario diferenciar el tipo de optimización a realizar (minimización o maximización), al punto que un objetivo puede ser maximizado, mientras que otro puede ser minimizado.

Se puede introducir el criterio de optimalidad Pareto, a partir del concepto de dominancia Pareto. Volviendo al gráfico, el punto A es único entre los demás: su vector de decisión $\boldsymbol{a}$ no está dominado por otro vector de decisión. Esto implica que $\boldsymbol{a}$ es óptimo en el sentido de que no puede mejorarse en un objetivo, sin degradar a otro. Tales soluciones se conocen como Pareto óptimas (también se utiliza el término «***no inferior***»).

DEFINICIÓN 1.2.7 (Optimalidad Pareto)**.** Dado un vector de decisión $\boldsymbol{x} \in \boldsymbol{X}_f$ y su correspondiente vector objetivo $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}) \in \boldsymbol{Y}_f$, se dice que $\boldsymbol{x}$ es no dominado respecto a un conjunto $\boldsymbol{A} \in \boldsymbol{X}_f$ si y solo si

$$\forall \boldsymbol{a} \in \boldsymbol{A} : (\boldsymbol{x} \succ \boldsymbol{a} \vee \boldsymbol{x} \sim \boldsymbol{a}) \tag{1.2.7}$$

En caso que $\boldsymbol{x}$ sea no dominado respecto a todo el conjunto $\boldsymbol{X}_f$, y solo en ese caso, se dice que $\boldsymbol{x}$ es una «***solución Pareto óptima***» ($\boldsymbol{x} \in \boldsymbol{X}_{true}$ – el conjunto Pareto Óptimo real). Mientras que la $\boldsymbol{y}$ correspondiente es parte del «***frente Pareto óptimo real*** $\boldsymbol{Y}_{true}$». Esto se define a continuación.

DEFINICIÓN 1.2.8 (Conjunto Pareto Óptimo y Frente Pareto Óptimo)**.** Dado el conjunto de vectores de decisión factibles $\boldsymbol{X}_f$. Se denomina $\boldsymbol{X}_{true}$ al conjunto de vectores de decisión no dominados que pertenecen a $\boldsymbol{X}_f$, es decir:

$$\boldsymbol{X}_{true} = \{\boldsymbol{x} \in \boldsymbol{X}_f | \boldsymbol{x} \text{ es no dominado con respecto a } \boldsymbol{X}_f\} \tag{1.2.8}$$

El conjunto $\boldsymbol{X}_{true}$ también es conocido como el «***conjunto Pareto óptimo***». Mientras que el conjunto correspondiente de vectores objetivo $\boldsymbol{Y}_{true} = \boldsymbol{f}(\boldsymbol{X}_{true})$ constituye el «***frente Pareto óptimo***».

Retomando el ejemplo de la Figura 1.2.1, los puntos de la curva (i.e. A, B) representan soluciones Pareto óptimas. Entre ellas son no comparables, o, en otras palabras, son indiferentes. Esto aclara aún más la diferencia de los MOP's con los SOP's, o problema de optimización de un único objetivo): en este caso no hay una única solución sino un conjunto de soluciones de compromiso. Ninguna de ellas se puede definir como «***mejor***» que las demás, a menos que se incluya alguna otra información que determine que un objetivo es más importante que los demás o que se fije un peso relativo entre los objetivos.

Como una ilustración adicional de los conceptos presentados, se discutirá a continuación un MOP específico, con una sola variable de decisión ($n = 1$), dos funciones objetivo ($k = 2$) y sin restricciones ($m = 0$).

EJEMPLO 1.2.2 (Ejemplo de un Problema de Optimización Multiobjetivo)**.**

$$\begin{aligned} \text{Minimizar} \quad & \boldsymbol{f}(x) = (f_1(x), f_2(x)) \\ \text{donde} \quad & f_1(x) = x^2, \\ & f_2(x) = (x-2)^2 \end{aligned} \tag{1.2.9}$$

Este par de funciones ha sido estudiado previamente en muchos otros trabajos,constituyéndose en un paradigma para la presentación de los principales conceptos de un MOP.

La Figura 1.2.2 presenta la gráfica de ambas funciones respecto a x. La misma sugiere que el conjunto Pareto óptimo es $\{x|x \geqslant 0 \text{ y } x \leqslant 2\}$. Esto es: $X_{true} = \{x \in \mathbb{R}|0 \leqslant x \leqslant 2\}$. La solución $x = 0$ es óptima respecto a f_1 pero no respecto a f_2. Cualquier solución que no se encuentre en el rango $[0, 2]$ no es miembro del conjunto Pareto óptimo, puesto que en dicho rango (X_{true}) existe siempre una solución que la domina.

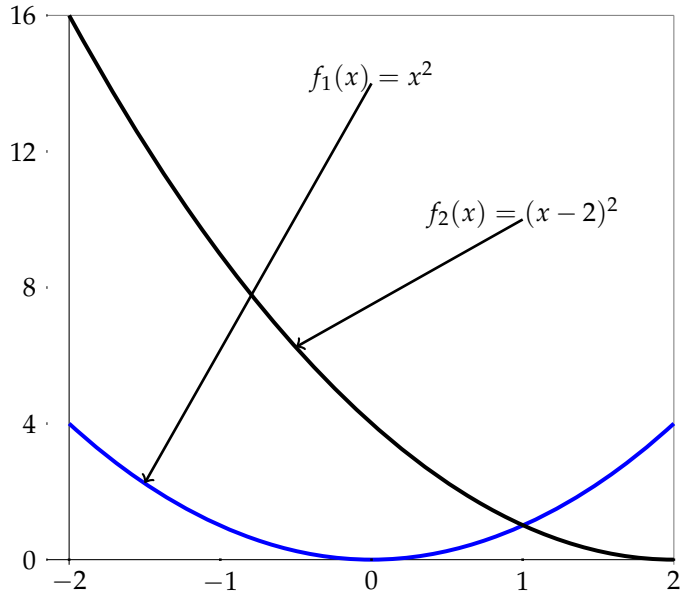


FIGURA 1.2.2: Valores de f_1 y f_2 en función de $x \in \boldsymbol{X}_f = (-\infty, \infty)$

La diferencia esencial entre la Figura 1.2.2 y la Figura 1.2.3 es que la primera representa los valores de las funciones f_1 y f_2 para diferentes valores de la variable independiente x. Mientras que la segunda representa los valores de la función f_1 graficados contra la función f_2, para el mismo valor de la variable independiente. Es decir, la Figura 1.2.3 es una gráfica en el espacio objetivo $\boldsymbol{Y}$, que muestra los vectores del MOP como puntos. Los vectores no dominados (que se muestran como puntos), representan el frente Pareto del MOP, por encima de la curva, se encuentra el espacio objetivo factible $\boldsymbol{Y}_f$.

§1.3. Búsqueda y Toma de Decisiones

Al resolver un MOP, se pueden distinguir 2 niveles de dificultad en el problema: la búsqueda y la toma de decisiones. El primer aspecto se refiere al proceso de optimización, durante el cual se explora el espacio de soluciones factibles buscando las soluciones Pareto óptimas. En esta fase, como ocurre en la optimización de objetivo único, el espacio de búsqueda puede ser lo su-

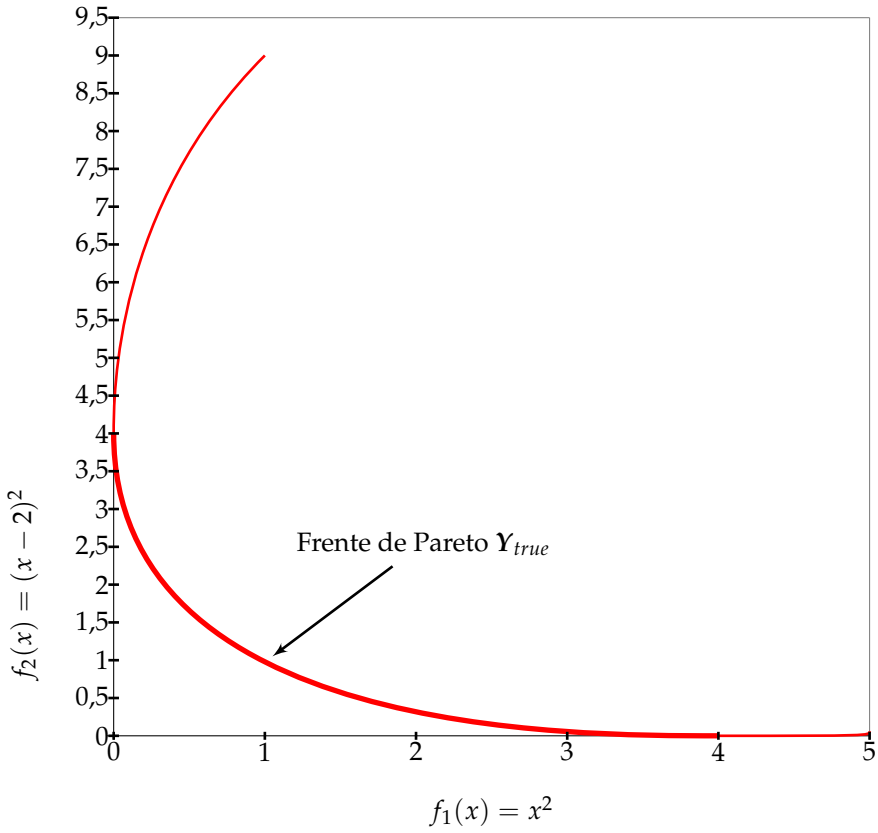


FIGURA 1.2.3: Frente Pareto del MOP

ficientemente grande y complejo como para impedir el uso de técnicas de optimización que den resultados exactos. El segundo aspecto es equivalente a seleccionar un conjunto de soluciones de compromiso que se utilizarán, del conjunto Pareto óptimo ya definido. Este paso, generalmente, lo realiza un ser humano y tiene que ver con cuestiones inherentes a las condiciones del problema y los recursos disponibles.

Dependiendo de la manera en que se combinan ambas fases (búsqueda y toma de decisiones) los métodos de optimización multiobjetivo pueden clasificarse en tres categorías diferentes:

a) **Métodos de toma de decisión previa a la búsqueda** (decidir, luego buscar): los objetivos del MOP se combinan en un único objetivo que implícitamente incluye información de preferencia obtenida del responsable de la toma de decisiones. Estos son métodos de toma de decisión a priori. Gran parte de las técnicas tradicionales para resolución de MOPs utilizan esta estrategia.

b) **Métodos de búsqueda previa a la toma de decisión** (buscar, luego decidir): se realiza la optimización sin incluir información de preferencia. El responsable de la toma de decisiones escoge del conjunto de soluciones obtenidas (que idealmente son todas del conjunto Pareto óptimo). A éstos se conoce como métodos de toma de decisión a posteriori.

c) **Métodos de toma de decisión durante la búsqueda** (buscar mientras se decide): éstos permiten que quien toma las decisiones pueda establecer preferencias durante un proceso de optimización interactivo. Luego de llevarse a cabo cada paso del proceso de optimización se presenta al responsable de decisiones los conflictos existentes y se le permite especificar sus preferencias o compromisos. De esta manera se desarrolla una búsqueda guiada. Estos métodos reciben el nombre de *progresivos* o *interactivos*.

La combinación de objetivos múltiples en un único objetivo a optimizar posee la ventaja de que a la función resultante se puede aplicar cualquiera de los métodos de optimización de un único objetivo ya disponibles en la literatura, sin mayores modificaciones. Sin embargo, tal combinación requiere conocimiento del dominio del problema que no siempre está disponible. Por ejemplo, en los problemas de diseño de diversas áreas de ingeniería, se realiza la búsqueda justamente para tener un mejor conocimiento del espacio de búsqueda del problema y las soluciones alternativas. Al realizar la búsqueda antes de la toma de decisión no se requiere este conocimiento, pero se impide que quien toma la decisión exprese previamente sus preferencias, lo que probablemente conduce a una reducción de la complejidad del espacio de búsqueda. Un problema con las técnicas de toma de decisión interactivas o a posteriori es la dificultad de presentar al responsable de la toma de decisiones las diferencias entre las soluciones de un MOP de muchas dimensiones. Aún así, la integración de la búsqueda y la toma de decisiones que proponen las técnicas interactivas parece una alternativa promisoria para aprovechar las ventajas de los dos primeros tipos de técnicas.

§1.4. MÉTODOS TRADICIONALES PARA RESOLVER MOPS

Numerosos métodos clásicos existen para el tratamiento de problemas de optimización multiobjetivo. Todos ellos se caracterizan por operar en dos fases. Primeramente generan, a partir del problema de objetivos múltiples, un problema de optimización de un solo objetivo (SOP). En la segunda fase aplican un método de optimización tradicional al SOP generado en la fase anterior y obtienen sus resultados. Ambas fases son independientes entre sí, i.e. una vez obtenido el SOP se puede resolverlo con cualquier técnica típica (hill climbing, algoritmos de búsqueda aleatoria, métodos numéricos, etc.).

Varios tipos de métodos se emplean para la reducción realizada en la primera fase, cada uno de ellos tiene sus propias ventajas y desventajas. En gran parte de ellos, la técnica consiste en disponer de las funciones objetivo de modo tal que se puedan unir en una única función parametrizada, mientras el proceso de optimización se aplica sobre esta única función. Generalmente, los parámetros de la función se varían de modo sistemático y se requieren múltiples corridas del procedimiento de optimización (2da. fase) para conseguir un conjunto que se aproxime al conjunto Pareto óptimo real.

Algunas técnicas representativas son «**Método de la suma ponderada**», «**Método de las ε-**

restricciones», «**Programación en base a objetivos**», y la «**Aproximación Min Max**». A continuación se presentarán, a modo de ejemplo, dos de los métodos más conocidos y de uso más extendido, exponiendo sus características más resaltantes. No discutiremos sobre los métodos empleados en la segunda fase, ya que son de conocimiento ampliamente difundido (búsqueda exhaustiva, programación lineal, optimización utilizando el concepto del gradiente, técnicas de inteligencia artificial, etc.) y al respecto existe ya una vasta bibliografía autorizada.

§1.4.1. Método de la Suma Ponderada

En este método se combinan las diferentes funciones objetivo en una sola función F, de la siguiente manera:

$$\begin{aligned} \text{Optimizar} \quad & F = \sum_{j=1}^{k} w_j f_j(\boldsymbol{x}) \\ \text{donde} \quad & \boldsymbol{x} \in \boldsymbol{X}_f, \\ & w_j \text{ es el peso usado para ponderar la } j\text{-ésima función objetivo.} \end{aligned} \tag{1.4.1}$$

Usualmente, y sin pérdida de generalidad, se escogen pesos fraccionales y distintos de cero, de manera que se cumpla $\sum_{j=1}^{k} w_j = 1, w_j > 0$.

El procedimiento es sencillo: se escoge una combinación de pesos (posiblemente aleatoria) y se optimiza la función F para obtener una solución óptima. Otras soluciones surgen a partir de optimizaciones realizadas sobre una combinación diferente de pesos.

En el presente contexto, optimizar implica maximizar todas las funciones objetivo, o minimizarlas. Esto es, o se maximiza o se minimiza todas las funciones $f_j(\boldsymbol{x})$; no se admite la maximización de algunos objetivos y la minimización de otros.

La interpretación de este método es la siguiente. Alterar el vector de pesos y optimizar la ecuación implica encontrar un hiperplano (una línea para el caso en que se tengan dos objetivos) con una orientación fija en el espacio de la función. La solución óptima es el punto donde un hiperplano con esta orientación tiene una tangente común con el espacio de búsqueda factible.

De esto último se deduce que este método no puede usarse para encontrar soluciones Pareto óptimas en problemas de optimización multiobjetivo que tienen un frente Pareto óptimo no convexo.

EJEMPLO 1.4.1. Dado el problema lineal biobjetivo

$$\begin{aligned} \text{Maximizar} \quad & \boldsymbol{z} = (-x_1 + x_2, x_1 + 2x_2) \\ \text{s.a.} \quad & \\ & -x_1 + 2x_2 \leqslant 8 \\ & x_1 + x_2 \leqslant 8 \\ & x_1 \leqslant 6 \\ & x_1, x_2 \geqslant 0 \end{aligned} \tag{1.4.2}$$

Se desea:

a) Resolverlo con el método de la suma ponderada tomando el vector de pesos $\lambda = (1, 1)$.

b) Generar (o aproximar) el conjunto eficiente en el espacio de decisiones por variación paramétrica.

c) Representar gráficamente la región factible en el espacio de decisiones y el conjunto eficiente.

SOLUCIÓN. a) En este caso se sustituye el objetivo vectorial

$$\boldsymbol{z} = (-x_1 + x_2, x_1 + 2x_2)$$

por una función objetivo escalar z, con pesos $\lambda = (1, 1)$, que toma la forma

$$z = 1 \times (-x_1 + x_2) + 1 \times (x_1 + 2x_2) = 3x_2.$$

Tenemos el Programa Lineal

$$P(1,1): \begin{cases} \text{máx}\, z = 3x_2 & \\ s.a. & \\ \qquad -x_1 + 2x_2 \leqslant 8 & \\ \qquad x_1 + x_2 \leqslant 8 & \\ \qquad x_1 \leqslant 6 & \\ \qquad x_1, x_2 \geqslant 0 & \end{cases}$$

Mediante el método del simplex, se obtiene la solución $x_1^* = 8/3 \approx 2{,}67$ y $x_2^* = 16/3 \approx 5{,}34$, que es eficiente, ya que el vector de pesos es $\lambda > \mathbf{0}$.

b) Consideramos el problema genérico de ponderaciones, cuya función objetivo es

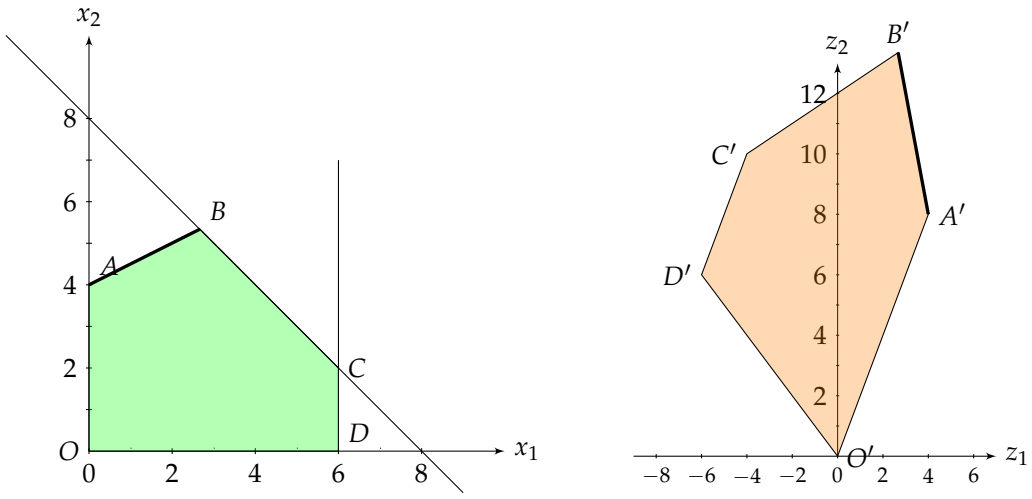
$$z = \lambda_1(-x_1 + x_2) + \lambda_2(x_1 + 2x_2).$$

Si vamos variando los pesos λ_1 y λ_2 de modo sistemático, al resolver los correspondientes programas lineales obtendremos el conjunto eficiente, o al menos una aproximación de él. La Tabla contiene, para valores escogidos de los pesos, las soluciones de los programas lineales asociados.

Pesos (λ_1, λ_2)	Solución (x_1^*, x_2^*)
(1,2)	(2.67,5.34)
(1,3)	(2.67,5.34)
(1,5)	(2.67,5.34)
(2,1)	(2.67,5.34)
(4,1)	(0,4)
(5,2)	(2.67,5.34)
(6,1)	(0,4)

Obsérvese que hemos ido tomando pesos estrictamente positivos para garantizar la eficiencia de las soluciones. A la vista de los resultados, podríamos afirmar que el conjunto eficiente $\mathcal{E}(F)$ es el segmento que une los puntos extremos $(0,4)$ y $(2{,}67, 5{,}34)$, que se comprueba que son puntos extremos eficientes de F.

c) Las figuras que siguen corresponden: la de la izquierda, a la región factible F en el espacio de decisiones $\mathbb{R}^2$, la de la derecha, a la región $Z = \boldsymbol{v}(F)$ en el espacio de objetivos o consecuencias $\mathbb{R}^2$. En ambas, se han marcado en negritas los correspondientes conjuntos eficientes, donde $A = (0,4), B = (2{,}67, 5{,}34), A' = (4,8)$ y $B' = (2.67, 13{,}35)$.



§1.4.2. Método de la ε-restricción

Para intentar remediar la dificultad arriba mencionada, se utiliza otra técnica. La misma consiste en construir un problema de optimización en el que todos los objetivos, excepto uno, se usan como restricciones ($k-1$ restricciones), mientras el sobrante, que puede escogerse aleatoriamente, se usa como función objetivo del SOP resultante. Este método es el que se ha escogido, hasta aho-

ra, para el tratamiento del problema de optimización de topologías de redes. La forma general de expresión del problema es la siguiente:

$$\begin{aligned} \text{Optimizar} \quad & f_h(\boldsymbol{x}) \\ \text{sujeto a} \quad & f_j(\boldsymbol{x}) \leqslant \varepsilon_j, \quad 1 \leqslant j \leqslant k, \quad j \neq k \\ & \boldsymbol{x} \in \boldsymbol{X}_f \end{aligned} \tag{1.4.3}$$

Para encontrar una solución Pareto óptima, se escoge un valor adecuado para ε_j de la j-ésima función objetivo ($j \neq h$). Luego se resuelve el problema de optimización con objetivo único. El procedimiento se repite, con diferentes valores de ε_j para hallar nuevas soluciones pertenecientes al frente Pareto óptimo. La dificultad principal que presenta este método reside en la necesidad de un conocimiento del rango apropiado de valores a asignar a ε_j para las $(k-1)$ funciones objetivo, lo que no resulta fácil en la práctica.

Resolviendo el Problema (1.4.2) usando $\varepsilon_2 = 10$, se obtiene por el método simlex, $x_1^* = 1, x_2^* = 4{,}5$ con $z_1^* = 3{,}5$ y $z_2^* = 10$.

§1.4.3. Discusión de los métodos clásicos

Aunque no se adaptan a la naturaleza del problema, la ventaja principal de los métodos tradicionales de primera fase que se mencionan en las secciones previas es que permite la utilización –en la segunda fase– de algoritmos de optimización para SOPs bien conocidos y de probada eficacia, aún cuando se trate de problemas reales, de gran tamaño. Para problemas de gran escala, muy pocas técnicas de optimización multiobjetivo reales se han presentado, a diferencia de técnicas de optimización de un solo objetivo que han existido desde hace bastante tiempo y han sido probadas en diferentes situaciones. Sin embargo, las secciones precedentes han mostrado que éstas técnicas no están exentas de dificultades que limitan su aplicación a la optimización de objetivos múltiples.

Todos los métodos aludidos, y aún otros propuestos en un contexto tradicional, cuentan con al menos una de las siguientes dificultades:

1. El algoritmo de optimización se debe aplicar varias veces para encontrar las soluciones Pareto óptimas múltiples. Como cada corrida es independiente de las demás, generalmente no se obtiene un efecto sinergético. Por tanto, delinear el frente Pareto óptimo resulta computacionalmente muy caro.

2. La mayoría de los algoritmos requieren conocimiento previo del problema a resolver y son sensibles a los pesos o niveles de demanda utilizados.

3. Algunos algoritmos son sensibles a la forma del frente Pareto (problemas con curvas no convexas).

4. La variación entre las diferentes soluciones encontradas depende de la eficiencia del optimizador de un solo objetivo. Podría darse el caso de encontrar siempre la misma solución o soluciones muy parecidas, en corridas múltiples.

5. Relacionado al punto anterior, tenemos que en problemas que involucren al azar o incertidumbre, los métodos clásicos no son necesariamente confiables.

6. Como los optimizadores de un solo objetivo no son eficientes en búsquedas de universos discretos, tampoco serán eficientes para optimizaciones multiobjetivo en espacios discretos.

Investigaciones recientes han demostrado que todas las dificultades arriba mencionadas pueden ser superadas con la utilización de un algoritmo evolutivo. Las características de éstos algoritmos hacen posible que:

a) se manejen espacios de búsqueda de casi cualquier tamaño y características; y

b) debido a su paralelismo inherente, se puedan generar múltiples soluciones en una sola corrida, permitiendo un efecto sinergético.

Con la afirmación previa, se introduce el criterio que se utilizará en el presente trabajo para lidiar con el problema multiobjetivo, es decir: se empleará algoritmos evolutivos. Es adecuado, entonces, dedicar la siguiente sección a la presentación de estos algoritmos.

§1.5. ALGORITMOS EVOLUTIVOS EN OPTIMIZACIÓN MULTIOBJETIVO

El término algoritmo evolutivo (EA[1] por sus siglas en inglés: Evolutionary Algorithm) se refiere a técnicas de búsqueda y optimización inspiradas en el modelo de la evolución propuesto por DARWIN en 1859, luego de sus viajes exploratorios.

En la naturaleza los individuos se caracterizan mediante cadenas de material genético que se denominan cromosomas. En los cromosomas se halla codificada toda la información relativa a un individuo y a sus tendencias. Cada elemento que conforma el cromosoma recibe el nombre de alelo. Cada individuo posee un nivel del adaptación al medio que lo dota de mayor capacidad de sobrevivencia y generación de descendencia. Tal nivel de adaptación está ligado a las características que están codificadas en sus cromosomas. Como el material genético puede transmitirse de padres a hijos al ocurrir el apareamiento, los hijos resultantes poseen cadenas de cromosomas parecidas a las de sus padres y combinan las características de los mismos. Por tanto si padres con buenas características se cruzan, posiblemente generarán hijos igualmente buenos o incluso mejores.

Para resolver un problema de búsqueda u optimización utilizando algoritmos evolutivos y los conceptos sugeridos, primero se representa como individuos de una población finita a un número dado de soluciones posibles del problema, a este proceso se denomina codificación. En la codificación de un individuo debe estar presente toda la información relevante al mismo y que se considera influye en la optimización o búsqueda. Generalmente la codificación de un individuo o

[1] Para una introducción más profunda y detallada a los EAs, referimos al lector a GOLDBERG (1989).

su cromosoma es una cadena de bits o de números enteros, dependiendo del problema que se desea resolver. En dicha cadena, el elemento que se encuentra en una posición dada recibe el nombre de allele.

Luego, se determina el nivel de aptitud o adaptación de cada individuo (fitness), dependiendo de la calidad de la solución que representa. Posteriormente los individuos existentes generan a otros individuos mediante los operadores genéticos como selección, cruzamiento y mutación. El operador de selección elige los padres que se cruzarán. La probabilidad de que un individuo sea escogido como padre y/o que sobreviva hasta la siguiente generación está ligada a su fitness o aptitud: a mayor fitness, mayor probabilidad de sobrevivencia y de tener descendientes, de la misma forma que ocurre en los procesos naturales.

Luego de escogerse los padres, se procede a la recombinación o cruzamiento de los mismos para obtener a la nueva generación. De esta manera, en cada nueva generación se tienen buenas probabilidades de que la población se componga de mejores individuos, ya que los hijos heredarán las características buenas de sus padres, y al combinarlas podrán ser aún mejores.

Por otro lado, durante la recombinación pueden ocurrir alteraciones (mutaciones) en la información genética de un individuo. Si tales alteraciones se producen para bien, originarán un individuo bueno con alto fitness y la alteración se transmitirá a los nuevos individuos; si el cambio no es benéfico, el individuo alterado tendrá un fitness bajo y poca o ninguna descendencia, con lo que la alteración prácticamente morirá con él. De esta manera, luego del curso de varias generaciones, la población habrá evolucionado hacia individuos genéticamente muy parecidos y que tienen un nivel de aptitud elevado, es decir, representan buenas soluciones al problema propuesto.

Los operadores descritos reciben el nombre de operadores de búsqueda u operadores genéticos. La reproducción enfoca la atención en los individuos con alto fitness, y de esta manera explota la información disponible sobre la adaptación del individuo al medio ambiente. La recombinación y la mutación perturban de alguna manera a los individuos y proveen así de heurísticas para la exploración del espacio de búsqueda. Por ello se dice que los EAs utilizan conceptos de explotación y exploración. A pesar de ser simplistas desde el punto de vista de la biología, estos algoritmos son suficientemente complejos como para proveer mecanismos de búsqueda robustos y que se adaptan a gran variedad de problemas.

Los orígenes de estos algoritmos se remontan a la década de los 50, cuando se empezaron a estudiar los primeros conceptos, pero los mismos no se formalizan sino hasta el trabajo de HOLLAND: HOLLAND, JOHN (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor

Aproximadamente desde mediados de la década del 70, en el ámbito de los EAs, se han introducido numerosos algoritmos que se pueden clasificar principalmente como Algoritmos Genéticos (GA), Programación Evolutiva (EP) y Estrategias de Evolución (ES), Sistemas clasificadores (CS) y Programación Genética (GP). A pesar de que el principio que los soporta es, en apariencia, muy simple, estos algoritmos han probado ser herramientas generales, robustas y po-

tentes. En la siguiente sub-sección se brinda una pequeña introducción a los algoritmos genéticos. Para mayor información relativa a los demás algoritmos evolutivos se refiere al lector a.

Para una introducción detallada de los conceptos que dan soporte a los algoritmos, se recomienda el libro introductorio de GOLDBERG.

§1.5.1. Algoritmos Genéticos

Los algoritmos genéticos se utilizan en varios ámbitos, principalmente para búsquedas y optimizaciones. En la práctica se implementa el algoritmo escogiendo una codificación para las posibles soluciones del problema. La codificación se realiza mediante cadenas de bits, números o caracteres para representar a los cromosomas. Luego, las operaciones de cruzamiento y mutación se aplican de manera muy sencilla mediante funciones de manipulación de valores de vectores. A pesar de que se han realizado numerosas investigaciones sobre codificación usando cadenas de longitud variable y aún otras estructuras, gran parte del trabajo con algoritmos genéticos se enfoca en cadenas de bits de longitud fija. Justamente el hecho de utilizar cadenas de longitud fija y la necesidad que existe de codificar las posibles soluciones son las dos características cruciales que diferencian a los algoritmos genéticos de la programación genética, que no posee representación de longitud fija y que normalmente no utiliza una codificación del problema y sus soluciones.

El ciclo de trabajo de un GA (también conocido como ciclo generacional) es generalmente el siguiente: calcular el fitness de los individuos en la población; crear una nueva población mediante selección, cruzamiento y mutación; y, finalmente, descartar la población vieja y seguir iterando utilizando la población recién creada. A cada iteración de este ciclo se la conoce como una generación. Cabe observar que no hay una razón teórica para que este sea el modelo de implementación. De hecho este comportamiento puntual no se observa como tal en la naturaleza. Sin embargo el modelo sigue siendo válido y conveniente.

La primera generación (generación 0) de este proceso, opera sobre una población de individuos generados al azar. A partir de allí, los operadores genéticos se aplican para mejorar a la población.

El operador de selección, según se ha apuntado, simula el proceso de selección natural en que el más fuerte tiene mayor capacidad de supervivencia. En el GA la capacidad de supervivencia de un individuo está ligada al valor numérico de la función objetivo o fitness. Este operador se aplica a cada iteración sobre una población de individuos de tamaño constante, con el objetivo de seleccionar individuos prometedores para generar la nueva población. Entre los individuos seleccionados pueden hallarse dos o más individuos idénticos, esto se debe a que los individuos con bajo fitness tienen poca probabilidad de ser elegidos, mientras que los de buen fitness son seleccionados con mayor frecuencia. El operador de selección puede aplicarse de maneras diversas. Unas veces se realiza la selección mediante torneos en que se escoge al azar un grupo de individuos y gana el torneo aquel con mejor fitness. La cantidad de individuos que se escogen para la competencia se fija de antemano y permanece constante en la implementación tradicional. Esta forma de selección recibe el nombre de selección por torneos y refleja de manera más adecuada el

proceso natural de selección. Otra forma, conocida como selección de ruleta, es fácilmente comprensible imaginando una ruleta en la que el número de partes en que se divide la misma es igual a la cantidad de individuos de la población, siendo el tamaño de cada parte proporcional al fitness de cada individuo. Es de esperar que al hacer girar la ruleta varias veces, se obtendrá mayor cantidad de individuos con alto fitness. Aunque pueden existir otras formas de realizar la selección, las dos anteriormente mencionadas son las más comunes en las implementaciones publicadas.

Una vez seleccionados los individuos se aplica a cada par de ellos el operador de cruzamiento. También el cruzamiento se puede realizar de maneras diferentes y aquí solo se discutirá una de ellas: el cruzamiento de 1 punto. En el mismo, se escoge aleatoriamente un punto de corte que se aplicará al par de cromosomas o codificación de los individuos seleccionados. Luego, los caracteres más significativos, a partir del punto de corte se conservan en sus posiciones relativas en el nuevo par de cromosomas, y los restantes son intercambiados de los cromosomas progenitores a los nuevos dos cromosomas obtenidos. La probabilidad de cruzamiento dada indica la probabilidad con que los progenitores se cruzarán y la probabilidad de generar clones; de esta forma, no todos los individuos seleccionados son destruidos por el cruzamiento.

El operado de cruzamiento representa una forma de búsqueda local, en las inmediaciones del espacio de búsqueda que rodea a los padres. Por su parte, el proceso de mutación es básicamente una búsqueda aleatoria. Se selecciona aleatoriamente una posición específica dentro de cromosoma del individuo a mutar, para luego cambiar el valor contenido en dicha posición.

Como en la naturaleza, la probabilidad de que ocurra una mutación es pequeña, en las condiciones de vida normales. En el GA se trata de representar lo mismo, con un valor de ocurrencia muy bajo para el operador de mutación.

§1.5.2. Algoritmos Evolutivos y optimización multiobjetivo

Las ventajas del uso de los algoritmos genéticos y evolutivos para resolver problemas de optimización multiobjetivo se notó rápidamente, ya que los algoritmos genéticos trabajan de antemano con poblaciones, lo que facilita la generación del conjunto de soluciones de Pareto (BROWN y SMITH, 2003; CRUZ-CORTÉS y COELLO, 2003; ISHIBUCHI y SHIBATA, 2003; LAW y KELTON, 1982).

SCHAFFER comenzó en 1985 a hacer algunos experimentos en esta área y propuso el algoritmo llamado VEGA (Vector Evaluated Genetic Algorithm), en el cual se utilizan subpoblaciones que optimizan cada objetivo por separado, es por esto que el concepto del óptimo de Pareto no está directamente incorporado en el mecanismo de selección de los GA. VEGA es eficiente y fácil de implementar, sin embargo no cuenta con un mecanismo explícito que mantenga la diversificación, además de que no produce necesariamente vectores no dominados.

En 1993, FONSECA y FLEMING propusieron el MOGA (Multi-Objective Genetic Algorithm), que consiste en un modelo en el que el lugar de cierto individuo corresponde al número de individuos en la población por los que está dominado. Es un algoritmo eficiente y relativamente sencillo, aunque los resultados dependen de un apropiado factor de compartición.

SRINIVAS y DEB propusieron en 1994 el NSGA (Nondominated Sorting Genetic Algorithm). Este algoritmo está basado en una serie de leyes de clasificación de los individuos. Los no dominados reciben un tipo de valores de engaño y luego se quitan de la población, proceso que se repite hasta que la población entera está clasificada. Para mantener la diversidad, los elementos clasificados son compartidos por sus valores de engaño. Este algoritmo es fácil de implementar, aunque parece que es muy sensible al valor del factor de compartición.

HORN y NAFPLIOTIS propusieron entre 1993 y 1994 el algoritmo llamado NPGA (Niched-Pareto Genetic Algorithm), que usa una selección basada en la dominancia de Pareto. Dos elementos se escogen aleatoriamente y se comparan contra un subconjunto de la población (del 10 % aproximadamente). Es sencillo de implementar y eficiente porque no aplica el ranking de Pareto a toda la población, por lo que se dice que tiene buena actuación. Sin embargo necesita otro factor además del factor de compartición, que es el tournament size.

A pesar de esta gran variedad, aún existen numerosas cuestiones que todavía permanecen abiertas. Entre ellas podemos citar:

a) ¿Qué algoritmos propuestos se desempeñan mejor en qué tipo de problemas?

b) ¿Cuáles son las ventajas y desventajas de un algoritmo particular, respecto a los demás?

c) ¿Qué medida de calidad se utilizará para afirmar que un algoritmo es efectivo en un caso dado?

d) ¿Las técnicas como niching, elitismo y restricciones de cruzamiento pueden mejorar el desempeño de los MOEAs?

e) ¿Cuáles son las técnicas más apropiadas para la paralelización de MOEAs?

f) ¿Cómo afecta el proceso de paralelización al desempeño de los algoritmos?

Gran parte de las dudas surgen del mismo hecho que los MOEAs se aplican a una cantidad enorme de problemas de diversos ámbitos, por lo que una generalización completa parece ser imposible.

Además, es importante evidenciar que algunas aplicaciones del mundo real presentan mayor dificultad para encontrar soluciones no inferiores (Pareto óptimas), con lo que la determinación del frente Pareto completo no es factible. Para estos casos, es necesario redefinir los objetivos de la resolución de MOPs considerando lo siguiente:

a) Se debe minimizar la distancia entre el frente Pareto verdadero y el frente conformado por las soluciones (no dominadas) halladas.

b) Se debe tener una buena distribución (generalmente esto equivale a decir una distribución uniforme) de las soluciones identificadas.

c) Es deseable encontrar soluciones bien diferenciadas unas de otras.

§1.6. CONCLUSIONES FINALES Y DELINEAMIENTO DE CAPÍTULOS POSTERIORES

Las cuestiones planteadas ayudan a delinear un marco de desarrollo. En el presente trabajo se pretende dar respuesta a las preguntas enunciadas, para el ámbito específico de resolver un problema de programación lineal multiobjetivo, usando algoritmos evolutivos.

Para ello, en primer término, en el siguiente capítulo se presentan más con más detalle, los conceptos necesarios de la teoría de algoritmos evolutivos, en particular de los algoritmos genéticos, y el teorema fundamental de estos algoritmos (Teorema de Los Esquemas).

En el capítulo 3 se discute sobre la aplicación de un algoritmo genético para resolver un problema multiobjetivo, en específico, el problema de la ruta más corta, estocástico multiobjetivo.

Por último se agrega un capítulo donde se presentan los resultados de los experimentos con el algoritmo creado para resolver el problema del capítulo 3.

CAPÍTULO 2
ALGORITMOS EVOLUTIVOS

§2.1. INTRODUCCIÓN

Como se mencionó en la sección 1.5 los Algoritmos Evolutivos son técnicas de búsqueda y optimización inspiradas en el modelo de la evolución. En este apartado desarrollaremos más a fondo los Algoritmos Genéticos (AGs). Los principios básicos de los Algoritmos Genéticos fueron establecidos por HOLLAND (1975), y se encuentran bien descritos en varios textos como GOLDBERG (1989); MICHALEWICZ (1992) y REEVES (1993).

En la naturaleza los individuos de una población compiten entre sí en la búsqueda de recursos tales como comida, agua y refugio. Incluso los miembros de una misma especie compiten a menudo en la búsqueda de un compañero. Aquellos individuos que tienen más éxito en sobrevivir y en atraer compañeros tienen mayor probabilidad de generar un gran número de descendientes. Por el contrario individuos poco dotados producirán un menor número de descendientes. Esto significa que los genes de los individuos mejor adaptados se propagarán en sucesivas generaciones hacia un número de individuos creciente. La combinación de buenas características provenientes de diferentes ancestros, puede a veces producir descendientes «***superindividuos***», cuya adaptación es mucho mayor que la de cualquiera de sus ancestros. De esta manera, las especies evolucionan logrando unas características cada vez mejor adaptadas al entorno en el que viven.

Los Algoritmos Evolutivos, o Computación Evolutiva, se pueden clasificar en tres categorías a saber; **Estrategias de Evolución**: Las bases de las Estrategias de Evolución fueron apuntadas en 1973 por Rechemberg en su obra "Evolutionsstrategie: Optimierung Technisher Systeme nach Prinzipien der Biologischen Evolution". Las dos Estrategias de Evolución más empleadas son la $(\lambda + \lambda)$-ES y la (λ, λ)-ES. En la primera de ellas un total de λ producen λ descendientes reduciéndose nuevamente la población a λ individuos (los padres de la siguiente generación) por selección de los mejores individuos. De esta manera los padres sobreviven hasta que son reemplazados por hijos mejores que ellos. En la $(\lambda + \lambda)$-ES la descendencia reemplaza directamente a los padres, sin hacer ningún tipo de comprobación, **Programación Evolutiva**: La Programación Evolutiva surge principalmente a raíz del trabajo "Artificial Intelligence Through Simulated Evolution"de Fogel, Owens y Walsh, publicado en 1966. En este caso los individuos, conocidos aquí como organismos, son máquinas de estado finito. Los organismos que mejor resuelven alguna de las funciones objetivo obtienen la oportunidad de reproducirse. Antes de producirse los cruces para

generar la descendencia se realiza una mutación sobre los padres. Y los Algoritmos Genéticos, que precisamente es la Técnica que vamos a usar y a detallar en el resto del Capítulo.

Los Algoritmos Genéticos usan una analogía directa con el comportamiento natural. Trabajan con una población de individuos, cada uno de los cuales representa una solución factible a un problema dado. A cada individuo se le asigna un valor ó puntuación, relacionado con la bondad de dicha solución. En la naturaleza esto equivaldría al grado de efectividad de un organismo para competir por unos determinados recursos. Cuanto mayor sea la adaptación de un individuo al problema, mayor será la probabilidad de que el mismo sea seleccionado para reproducirse, cruzando su material genético con otro individuo seleccionado de igual forma. Este cruce producirá nuevos individuos –descendientes de los anteriores– los cuales comparten algunas de las características de sus padres. Cuanto menor sea la adaptación de un individuo, menor será la probabilidad de que dicho individuo sea seleccionado para la reproducción, y por tanto de que su material genético se propague en sucesivas generaciones.

De esta manera se produce una nueva población de posibles soluciones, la cual reemplaza a la anterior y verifica la interesante propiedad de que contiene una mayor proporción de buenas características en comparación con la población anterior. Así a lo largo de las generaciones las buenas características se propagan a través de la población. Favoreciendo el cruce de los individuos mejor adaptados, van siendo exploradas las áreas más prometedoras del espacio de búsqueda. Si el Algoritmo Genético ha sido bien diseñado, la población convergerá hacia una solución óptima del problema. El poder de los Algoritmos Genéticos proviene del hecho de que se trata de una técnica robusta, y pueden tratar con éxito una gran variedad de problemas provenientes de diferentes áreas, incluyendo aquellos en los que otros métodos encuentran dificultades. Si bien no se garantiza que el Algoritmo Genético encuentre la solución óptima del problema, existe evidencia empírica de que se encuentran soluciones de un nivel aceptable, en un tiempo competitivo con el resto de algoritmos de optimización combinatoria. En el caso de que existan técnicas especializadas para resolver un determinado problema, lo más probable es que superen al Algoritmo Genético, tanto en rapidez como en eficacia. El gran campo de aplicación de los Algoritmos Genéticos se relaciona con aquellos problemas para los cuales no existen técnicas especializadas. Incluso en el caso en que dichas técnicas existan, y funcionen bien, pueden efectuarse mejoras de las mismas hibridándolas con los Algoritmos Genéticos.

La estructura de este capítulo es como sigue: en la siguiente sección se introduce por medio de un ejemplo el denominado Algoritmo Genético Simple, también conocido como Algoritmo Genético Canónico, para a continuación, mostrar distintas extensiones y modificaciones del mismo, relativas a los operadores de selección, cruce y mutación. En la siguiente sección nos preguntamos el motivo por el cual funcionan los Algoritmos Genéticos, estudiando el teorema de los esquemas.

§2.2. EL ALGORITMO GENÉTICO SIMPLE (AGS)

El Algoritmo Genético Simple, también denominado Canónico, se representa en el Pseudocódigo 2.2.1. Como se verá a continuación, se necesita una codificación o representación del problema, que resulte adecuada al mismo. Además se requiere una función de ajuste ó adaptación al problema, la cual asigna un número real a cada posible solución codificada. Durante la ejecución del algoritmo, los padres deben ser seleccionados para la reproducción, a continuación dichos padres seleccionados se cruzarán generando dos hijos, sobre cada uno de los cuales actuará un operador de mutación. El resultado de la combinación de las anteriores funciones será un conjunto de individuos (posibles soluciones al problema), los cuales en la evolución del Algoritmo Genético formarán parte de la siguiente población.

PSEUDOCÓDIGO 2.2.1: Pseudocódigo del Algoritmo Genético Básico

```
Comenzar
    t=0//comenzar en un tiempo inicial;
    Inicializar Población P(t) // inicializar una población de individuos generados al azar;
    Evaluar P(t) // evaluar el fitness de todos los individuos de la población inicial;
    Mientras no se cumpla la condición de parada: hacer
        Comenzar
            t = t + 1 // incrementar el contador de tiempo;
            P'= Seleccionar Padres P(t) // seleccionar una poblacion para generar
            descendientes;
            Recombinar P'(t) // recombinar los «genes» del grupo de padres seleccionados;
            Mutar P'(t) // perturbar la población generada de manera estocástica;
            Evaluar P'(t) // calcular fitness de la población recién creada;
            P = Sobrevivientes P, P'(t)// Seleccionar sobrevivientes para la siguiente
            generación;
        Terminar
Terminar
```

§2.2.1. Codificación

Se supone que los individuos (posibles soluciones del problema), pueden representarse como un conjunto de parámetros (que denominaremos genes), los cuales agrupados forman una ristra de valores (a menudo referida como cromosoma). Si bien el alfabeto utilizado para representar los individuos no debe necesariamente estar constituido por el $\{0, 1\}$, buena parte de la teoría en la que se fundamentan los Algoritmos Genéticos utiliza dicho alfabeto.

En términos biológicos, el conjunto de parámetros representando un cromosoma particular se denomina «***fenotipo***». El fenotipo contiene la información requerida para construir un organismo, el cual se refiere como genotipo. Los mismos términos se utilizan en el campo de los Algoritmos Genéticos. La adaptación al problema de un individuo depende de la evaluación del genotipo. Esta

última puede inferirse a partir del fenotipo, es decir puede ser computada a partir del cromosoma, usando la función de evaluación.

La «***función de adaptación***» debe ser diseñada para cada problema de manera específica. Dado un cromosoma particular, la función de adaptación le asigna un número real, que se supone refleja el nivel de adaptación al problema del individuo representado por el cromosoma.

Durante la «***fase reproductiva***» se seleccionan los individuos de la población para cruzarse y producir descendientes, que constituirán, una vez mutados, la siguiente generación de individuos. La «***selección de padres***» se efectúa al azar usando un procedimiento que favorezca a los individuos mejor adaptados, ya que a cada individuo se le asigna una probabilidad de ser seleccionado que es proporcional a su función de adaptación. Este procedimiento se dice que está basado en la ruleta sesgada. Según dicho esquema, los individuos bien adaptados se escogerán probablemente varias veces por generación, mientras que los pobremente adaptados al problema, no se escogerán más que de vez en cuando.

Una vez seleccionados dos padres, sus cromosomas se combinan, utilizando habitualmente los «***operadores de cruce y mutación***». Las formas básicas de dichos operadores se describen a continuación.

El «***operador de cruce***», coge dos padres seleccionados y corta sus ristras de cromosomas en una posición escogida al azar, para producir dos sub-ristras iniciales y dos sub-ristras finales. Después se intercambian las sub-ristras finales, produciéndose dos nuevos cromosomas completos (véase la Figura 2.2.1). Ambos descendientes heredan genes de cada uno de los padres. Este operador se conoce como operador de cruce basado en un punto. Habitualmente el operador de cruce no se aplica a todos

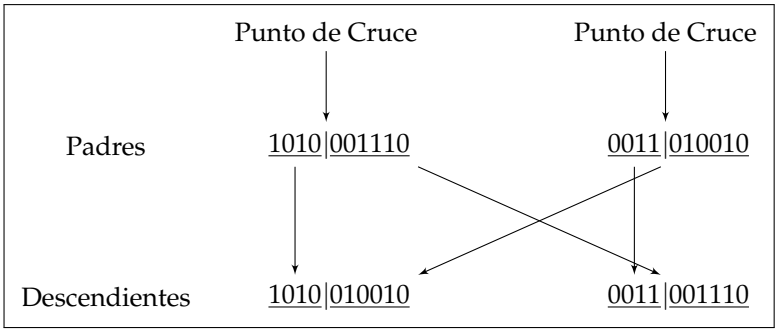


FIGURA 2.2.1: Operador de cruce basado en un punto

los pares de individuos que han sido seleccionados para emparejarse, sino que se aplica de manera aleatoria, normalmente con una probabilidad comprendida entre 0,5 y 1,0. En el caso en que el operador de cruce no se aplique, la descendencia se obtiene simplemente duplicando los padres.

El «***operador de mutación***» se aplica a cada hijo de manera individual, y consiste en la al-

teración aleatoria (normalmente con probabilidad pequeña) de cada gen componente del cromosoma. La Figura 2.2.2 muestra la mutación del quinto gen del cromosoma.

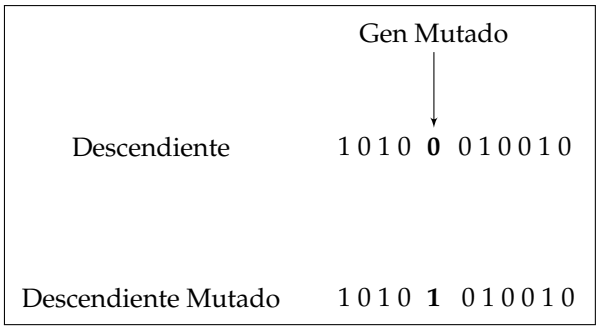


FIGURA 2.2.2: Operador de mutación

TABLA 2.2.1: Población inicial de la simulación efectuada a mano correspondiente al Algoritmo Genético Simple

	Población inicial (fenotipos)	x valor genotipo	$f(x)$ valor (función adaptación)	$f(x)/\sum f(x)$ (probabilidad selección)	Probabilidad de selección acumulada
1	01101	13	169	0.14	0.14
2	11000	24	576	0.49	0.63
3	01000	8	64	0.06	0.69
4	10011	19	361	0.31	1.00
Suma			1170		
Media			293		
Mejor			576		

Si bien puede en principio pensarse que el operador de cruce es más importante que el operador de mutación, ya que proporciona una exploración rápida del espacio de búsqueda, éste último asegura que ningún punto del espacio de búsqueda tenga probabilidad cero de ser examinado, y es de capital importancia para asegurar la convergencia de los Algoritmos Genéticos. Para criterios prácticos, es muy útil la definición de convergencia introducida en este campo por DE-JONG,1975 en su tesis doctoral.

Si el Algoritmo Genético ha sido correctamente implementado, la población evolucionará a lo largo de las generaciones sucesivas de tal manera que la adaptación media extendida a todos los individuos de la población, así como la adaptación del mejor individuo se irán incrementando hacia el óptimo global. El concepto de convergencia está relacionado con la progresión hacia la uniformidad: un gen ha convergido cuando al menos el 95 % de los individuos de la población comparten el mismo valor para dicho gen. Se dice que la población converge cuando todos los genes han convergido. Se puede generalizar dicha definición al caso en que al menos un β % de los

individuos de la población hayan convergido. A medida que el número de generaciones aumenta, es más probable que la adaptación media se aproxime a la del mejor individuo.

§2.2.2. Ejemplo

Como ilustración de los diferentes componentes del Algoritmo Genético Simple, supongamos el problema adaptado de GOLDBERG (1989) de encontrar el máximo de la función $f(x) = x^2$ sobre los enteros $\{1, 2, \ldots, 32\}$. Evidentemente para lograr dicho óptimo, bastaría actuar por búsqueda exhaustiva, dada la baja cardinalidad del espacio de búsqueda. Se trata por tanto de un mero ejemplo con el que pretendemos ilustrar el comportamiento del algoritmo anteriormente descrito. Consultando el Pseudocódigo 2.2.1, vemos que el primer paso a efectuar consiste en determinar el tamaño de la población inicial, para a continuación obtener dicha población al azar y computar la función de evaluación de cada uno de sus individuos.

Suponiendo que el alfabeto utilizado para codificar los individuos esté constituido por $\{0, 1\}$, necesitaremos ristras de longitud 5 para representar los 32 puntos del espacio de búsqueda.

En la Tabla 2.2.1, hemos representado los 4 individuos que constituyen la población inicial, junto con su función de adaptación al problema, así como la probabilidad de que cada uno de dichos individuos sea seleccionado según el modelo de ruleta sesgada para emparejarse.

Volviendo a consultar el Pseudocódigo 2.2.1, vemos que el siguiente paso consiste en la selección de 2 parejas de individuos. Para ello es suficiente, con obtener 4 números reales provenientes de una distribución de probabilidad uniforme en el intervalo $\{0, 1\}$, y compararlos con la última columna de la Tabla 2.2.1. Así por ejemplo, supongamos que dichos 4 números hayan sido:

TABLA 2.2.2: Población en el tiempo 1

Emparejamiento de los individuos seleccionados	Punto de cruce	Descendientes	Nueva población descendientes mutados	x valor genotipo	$f(x)$ función adaptación
11000	2	11011	11011	27	729
10011	2	10000	10000	16	256
01101	3	01100	11100	28	784
11000	3	11101	11101	29	841
Suma				2610	
Media				652.5	
Mejor				841	

0,58; 0,84; 0,11 y 0,43. Esto significa que los individuos seleccionados para el cruce han sido: el individuo 2 junto con el individuo 4, así como el individuo 1 junto con el individuo 3. Para seguir con el Algoritmo Genético Simple, necesitamos determinar la probabilidad de cruce, p_c. Supongamos que se fije en $p_c = 0{,}8$. Valiéndonos al igual que antes de, 2 en este caso, números provenientes de la distribución uniforme, determinaremos si los emparejamientos anteriores se llevan a cabo. Admitamos, por ejemplo, que los dos números extraídos sean menores que 0,8,

decidiéndose por tanto efectuar el cruce entre las dos parejas. Para ello escogeremos un número al azar entre 1 y $l-1$ (siendo l la longitud de la ristra utilizada para representar el individuo). Notése que la restricción impuesta al escoger el número entre 1 y $l-1$, y no l, se realiza con la finalidad de que los descendientes no coincidan con los padres. Supongamos, tal y como se indica en la Tabla 2.2.2, que los puntos de cruce resulten ser 2 y 3. De esta manera obtendríamos los 4 descendientes descritos en la tercera columna de la Tabla 2.2.2. A continuación siguiendo el pseudocódigo de la Figura 6.1, mutaríamos con una probabilidad, p_m, cercana a cero, cada uno de los bit de las cuatro ristras de individuos. En este caso suponemos que el único bit mutado corresponde al primer gen del tercer individuo. En las dos últimas columnas se pueden consultar los valores de los individuos, así como las funciones de adaptación correspondientes. Como puede observarse, tanto el mejor individuo como la función de adaptación media han mejorado sustancialmente al compararlos con los resultados de la Tabla 6.1.

§2.3. Extensiones y Modificaciones del Algoritmo Genético Simple

En este apartado se introducirán algunas extensiones y modificaciones del Algoritmo Genético Simple. Se comenzará dando un pseudocódigo para un Algoritmo Genético Abstracto, para luego continuación formalizar matemáticamente cada uno de los elementos integrantes del Algoritmo, así como las extensiones y modificaciones que se vayan presentando.

Una versión del Algoritmo Genético Abstracto (AGA), puede ser como el del Pseudocódigo 2.3.1. Ya que el principal dominio de aplicación de los Algoritmos Genéticos lo constituye la optimización de funciones, se introducen algunos conceptos básicos que se usarán a lo largo de este Capítulo. Dado un dominio finito D y una función $f : D \to \mathbb{R}$, el problema de la «***optimización de la función***» f se refiere a encontrar el mejor valor de la función f en el dominio D. Se trata por tanto de encontrar $x \in D$ para el cual $f(x) \leqslant f(y), \quad \forall y \in D$.

Ya que $\text{máx}\{f(x)\} = -\text{mín}\{f(x)\}$ la restricción al problema de minimización no supone ninguna pérdida de generalización. En general, la tarea de optimización se complica debido a la existencia de óptimos locales (mínimos locales en nuestro caso).

La función f tiene un mínimo local en $\hat{x} \in D$ si se verifica la siguiente condición: $\exists E(x)$, entorno de x, tal que si $y \in E(x) \Rightarrow f(x) \leqslant f(y)$:

Diremos que $e : D \to l$ donde $l \geqslant \lg_{\|s\|} \|D\|$ constituye una «***codificación***», siendo la finalidad de la misma el representar los elementos de D por medio de ristras de elementos de S. A S se le denomina «***alfabeto***», mientras que S^l constituye el «***espacio de búsqueda***». A la función $f(x) = g(e(x))$ se le denomina función objetivo.

PSEUDOCÓDIGO 2.3.1: Pseudocódigo del Algoritmo Genético Abstracto

```
Comenzar
    Obtener la poblacion inicial al azar;
    Mientras no se cumpla la condición de parada: hacer
        Comenzar
            Seleccionar padres de la poblacion;
            Producir hijos a partir de los padres seleccionados;
            Mutar los individuos hijos;
            Extender la poblacion añadiendo los hijos;
            Reducir la poblacion extendida
        Terminar
Terminar
```

§2.4. POBLACIÓN

§2.4.1. Tamaño de la población

Una cuestión que uno puede plantearse es la relacionada con el tamaño idóneo de la población. Parece intuitivo que las poblaciones pequeñas corren el riesgo de no cubrir adecuadamente el espacio de búsqueda, mientras que el trabajar con poblaciones de gran tamaño puede acarrear problemas relacionados con el excesivo costo computacional.

GOLDBERG en 1989 efectuó un estudio teórico, obteniendo como conclusión que el tamaño óptimo de la población para ristras de longitud l, con codificación binaria, crece exponencialmente con el tamaño de la ristra.

Este resultado traería como consecuencia que la aplicabilidad de los Algoritmos Genéticos en problemas reales sería muy limitada, ya que resultarían no competitivos con otros métodos de optimización combinatoria. Alander (1992), basándose en evidencia empírica sugiere que un tamaño de población comprendida entre l y $2l$ es suficiente para atacar con éxito los problemas.

§2.4.2. Población inicial

Habitualmente la población inicial se escoge generando ristras al azar, pudiendo contener cada gen uno de los posibles valores del alfabeto con probabilidad uniforme. Nos podríamos preguntar que es lo que sucedería si los individuos de la población inicial se obtuviesen como resultado de alguna técnica heurística o de optimización local. En los pocos trabajos que existen sobre este aspecto, se constata que esta inicialización no aleatoria de la población inicial, puede acelerar la convergencia del Algoritmo Genético. Sin embargo en algunos casos la desventaja resulta ser la prematura convergencia del algoritmo, queriendo indicar con ésto la convergencia hacia óptimos locales.

§2.5. FUNCIÓN OBJETIVO

Dos aspectos que resultan cruciales en el comportamiento de los Algoritmos Genéticos son la determinación de una adecuada función de adaptación o función objetivo, así como la codificación utilizada.

Idealmente nos interesaría construir funciones objetivo con «***ciertas regularidades***», es decir funciones objetivo que verifiquen que para dos individuos que se encuentren cercanos en el espacio de búsqueda, sus respectivos valores en las funciones objetivo sean similares. Por otra parte una dificultad en el comportamiento del Algoritmo Genético puede ser la existencia de gran cantidad de óptimos locales, así como el hecho de que el óptimo global se encuentre muy aislado.

La regla general para construir una buena función objetivo es que ésta debe reflejar el valor del individuo de una manera «***real***», pero en muchos problemas de optimización combinatoria, donde existen gran cantidad de restricciones, buena parte de los puntos del espacio de búsqueda representan individuos no válidos.

Para este planteamiento en el que los individuos están sometidos a restricciones, se han propuesto varias soluciones. La primera sería la que podríamos denominar absolutista, en la que aquellos individuos que no verifican las restricciones, no son considerados como tales, y se siguen efectuando cruces y mutaciones hasta obtener individuos válidos, o bien a dichos individuos se les asigna una función objetivo igual a cero.

Otra posibilidad consiste en reconstruir aquellos individuos que no verifican las restricciones. Dicha reconstrucción suele llevarse a cabo por medio de un nuevo operador que se acostumbra a denominar reparador.

Otro enfoque está basado en la penalización de la función objetivo. La idea general consiste en dividir la función objetivo del individuo por una cantidad (la penalización) que guarda relación con las restricciones que dicho individuo viola. Dicha cantidad puede simplemente tener en cuenta el número de restricciones violadas ó bien el denominado costo esperado de reconstrucción, es decir el coste asociado a la conversión de dicho individuo en otro que no viole ninguna restricción.

Otra técnica que se ha venido utilizando en el caso en que la computación de la función objetivo sea muy compleja es la denominada evaluación aproximada de la función objetivo. En algunos casos la obtención de n funciones objetivo aproximadas puede resultar mejor que la evaluación exacta de una única función objetivo (supuesto el caso de que la evaluación aproximada resulta como mínimo n veces más rápida que la evaluación exacta).

Un problema habitual en las ejecuciones de los Algoritmos Genéticos surge debido a la velocidad con la que el algoritmo converge. En algunos casos la convergencia es muy rápida, lo que suele denominarse convergencia prematura, en la cual el algoritmo converge hacia óptimos locales, mientras que en otros casos el problema es justo el contrario, es decir se produce una convergencia lenta del algoritmo. Una posible solución a estos problemas pasa por efectuar transformaciones en la función objetivo. El problema de la convergencia prematura, surge a menudo cuando la selección de individuos se realiza de manera proporcional a su función objetivo. En tal

caso, pueden existir individuos con una adaptación al problema muy superior al resto, que a medida que avanza el algoritmo «***dominan***» a la población. Por medio de una transformación de la función objetivo, en este caso una comprensión del rango de variación de la función objetivo, se pretende que dichos «***superindividuos***» no lleguen a dominar a la población.

El problema de la lenta convergencia del algoritmo, se resolvería de manera análoga, pero en este caso efectuando una expansión del rango de la función objetivo.

La idea de especies de organismos, ha sido imitada en el diseño de los Algoritmos Genéticos en un método propuesto por Goldberg y Richardson (1987), utilizando una modificación de la función objetivo de cada individuo, de tal manera que individuos que estén muy cercanos entre sí devalúen su función objetivo, con objeto de que la población gane en diversidad.

Por ejemplo, si denotamos por $d(I_t^j, I_t^i)$ a la distancia de Hamming entre los individuos I_t^j e I_t^i y por $K \in \mathbb{R}^+$ a un parámetro, podemos definir la siguiente función:

$$h(d(I_t^j, I_t^i)) = \begin{cases} k - d(I_t^j, I_t^i) & \text{si } d(I_t^j, I_t^i) < k \\ 0 & \text{si } d(I_t^j, I_t^i) \geqslant k \end{cases} \tag{2.5.1}$$

A continuación para cada individuo I_t^j, definimos $\sigma_j^t = \sum_{i \neq j} h(d(I_t^j, I_t^i))$, valor que utilizaremos para devaluar la función objetivo del individuo en cuestión. Es decir, $g^*(I_t^j) = g(I_t^j)/\sigma_j^t$. De esta manera aquellos individuos que están cercanos entre sí verán devaluada la probabilidad de ser seleccionados como padres, aumentándose la probabilidad de los individuos que se encuentran más aislados.

§2.6. SELECCIÓN

La función de selección de padres más utilizada es la denominada «***función de selección proporcional a la función objetivo***», en la cual cada individuo tiene una probabilidad de ser seleccionado como padre que es proporcional al valor de su función objetivo.

Denotando por $p_{j,t}^{prop}$ la probabilidad de que el individuo I_t^j sea seleccionado como padre, se tiene que:

$$p_{j,t}^{prop} = \frac{g(I_t^j)}{\sum_{j=1}^{\lambda} g(I_t^j)} \tag{2.6.1}$$

Una de las maneras de superar el problema relacionado con la rápida convergencia proveniente de los superindividuos, que surge al aplicar la anterior función de selección, es el efectuar la ***selección proporcional al rango del individuo***, con lo cual se produce una repartición más uniforme de la probabilidad de selección.

Si denotamos por rango$(g(I_t^j))$ el rango de la función objetivo del individuo I_t^j cuando los individuos de la población han sido ordenados de menor a mayor (es decir el peor individuo tiene rango 1, mientras que el individuo con mejor función objetivo tiene rango λ), y sea $p_{j,t}^{rango}$ la

probabilidad de que el individuo I_t^j sea seleccionado como padre cuando la selección se efectúa proporcionalmente al rango del individuo, se tiene que

$$p_{j,t}^{rango} = \frac{\text{rango}(g(I_t^j))}{\lambda(\lambda+1)/2} \tag{2.6.2}$$

La suma de los rangos, $\lambda(\lambda+1)/2$, constituye la constante de normalización.

La ***selección por torneo***, constituye un procedimiento de selección de padres muy extendido y en el cual la idea consiste en escoger al azar un número de individuos de la población, tamaño del torneo, (con o sin reemplazamiento), seleccionar el mejor individuo de este grupo, y repetir el proceso hasta que el número de individuos seleccionados coincida con el tamaño de la población. Habitualmente el tamaño del torneo es 2, y en tal caso se ha utilizado una versión probabilística en la cual se permite la selección de individuos sin que necesariamente sean los mejores.

Una posible clasificación de procedimientos de selección de padres consistiría en: ***métodos de selección dinámicos***, en los cuales los probabilidades de selección varían de generación a generación, (por ejemplo la selección proporcional a la función objetivo), frente a ***métodos de selección estáticos***, en los cuales dichas probabilidades permanecen constantes (por ejemplo la selección basada en rangos).

§2.7. CRUCE

El Algoritmo Genético Canónico descrito anteriormente, utiliza el ***cruce basado en un punto***, en el cual los dos individuos seleccionados para jugar el papel de padres, son recombinados por medio de la selección de un punto de corte, para posteriormente intercambiar las secciones que se encuentran a la derecha de dicho punto.

Se han investigado otros operadores de cruce, habitualmente teniendo en cuenta más de un punto de cruce. DE-JONG investigó el comportamiento del ***operador de cruce basado en múltiples puntos***, concluyendo que el cruce basado en dos puntos, representaba una mejora mientras que añadir más puntos de cruce no beneficiaba el comportamiento del algoritmo. La ventaja de tener más de un punto de cruce radica en que el espacio de búsqueda puede ser explorado más fácilmente, siendo la principal desventaja el hecho de aumentar la probabilidad de ruptura de buenos esquemas.

§2.8. MUTACIÓN

La mutación se considera un operador básico, que proporciona un pequeño elemento de aleatoridad en la vecindad (entorno) de los individuos de la población. Si bien se admite que el operador de cruce es el responsable de efectuar la búsqueda a lo largo del espacio de posibles soluciones, también parece desprenderse de los experimentos efectuados por varios investigadores que el operador de mutación va ganando en importancia a medida que la población de individuos va convergiendo.

Si bien en la mayoría de las implementaciones de Algoritmos Genéticos se asume que tanto la probabilidad de cruce como la de mutación permanecen constantes, algunos autores han obtenido mejores resultados experimentales modificando la probabilidad de mutación a medida que aumenta el número de iteraciones.

§2.9. ¿POR QUÉ FUNCIONAN?

En este apartado se tratarán algunas cuestiones relacionadas con el por qué «***funcionan***» los Algoritmos Genéticos. El significado de la palabra «***funcionan***» se refiere al hecho de ser capaces de encontrar el óptimo de la función durante el proceso de búsqueda.

Se estudiará en primer lugar el denominado **Teorema De Los Esquemas**, (GOLDBERG, 1989), adaptando la notación, para presentarlo en relación con distribuciones binomiales, para a continuación introducir la denominada ***Hipótesis de bloques***, y finalizar haciendo referencia a algunos trabajos relacionados con la convergencia de los Algoritmos Genéticos.

§2.9.1. Teorema de los esquemas

En lo que sigue se desarrollará el denominado Teorema de los esquemas para el caso del Algoritmo Genético Canónico, utilizando además un alfabeto binario. Dicho teorema utiliza el concepto de esquema, que introducimos a continuación.

Supongamos un alfabeto binario $S = \{0, 1\}$. Para construir un esquema necesitamos ampliar el alfabeto anterior por medio del símbolo $*$. Un ***esquema*** será cualquier ristra de caracteres formada a partir de elementos del alfabeto ampliado $S' = \{0, 1, *\}$. Si la longitud de la ristra es l , el número de esquemas erxistentes es 3^l, ya que cada posición puede ocuparse por cualquier de los tres elementos del alfabeto extendido $'$. Así por ejemplo considerando ristras de longitud 4, tenemos que el esquema $Q = (*1*0)$ se empareja con los cromosomas

(0100),(0110),(1100),(1110).

Desde un punto de vista geométrico un esquema equivale a un hiperplano en el espacio de búsqueda.

Para desarrollar el Teorema de los esquemas necesitamos utilizar los conceptos de orden y longitud de un esquema, que se definen a continuación.

El orden del esquema Q, se denota por $\mathcal{O}(Q)$, y se refiere al número de posiciones ocupadas por $0 - s$, ó $1 - s$ dentro del esquema. Se trata por tanto del número de posiciones fijas en el esquema. En otras palabras se trata de la longitud de la ristra menos el número de símbolos $*$. Por ejemplo, los tres esquemas siguientes, $Q_1 = (*01*)$,$Q_2 = (***1)$ y $Q_3 = (110*)$, tienen los siguientes ordenes: $\mathcal{O}(Q_1) = 2$, $\mathcal{O}(Q_2) = 1$ y $\mathcal{O}(Q_3) = 3$.

La longitud del esquema Q, se denota por medio de $\delta(Q)$, y se define como la distancia entre la primera y la última posiciones fijas de la ristra. Dicho concepto define en cierto modo la

compacidad de la información contenida en el esquema. Así por ejemplo: $\delta(Q_1) = 3-2 = 1$, $\delta(Q_2) = 4-4 = 0$ y $\delta(Q_3) = 3-1 = 2$.

Denotaremos por:

λ, el tamaño de la población que se considerará constante a lo largo de la evolución,

p_c, la probabilidad de cruce,

p_m, la probabilidad de mutación.

Por otra parte, $\xi(Q,t)$ denotará el número de ristras que en una determinada población en el tiempo t, se asocian con el esquema Q. Otra propiedad del esquema es su adaptación en el tiempo t, la cual se denotará por $\text{eval}(Q,t)$ y se define como la media de la función objetivo de todas las ristras que en la población se asocian con el esquema Q.

Sea $\overline{F(t)}$, la media de la función objetivo extendida a toda la población en el tiempo t. Es decir $\overline{F(t)} = F(t)/\lambda$, siendo $F(t)$ la suma de las funciones objetivo de todas los individuos en la población en el tiempo t.

Con la notación introducida hasta el momento, podemos enunciar el Teorema de los esquemas de la siguiente manera.

TEOREMA 2.9.1 (Teorema de los Esquemas). *Si denotamos por $E_{sel,cru,mut}(\xi(Q,t+1))$, el número esperado de individuos que se asocian con el esquema Q, en el tiempo $t+1$, después de efectuar la selección, el cruce y la mutación en un Algoritmo Genético Canónico, se tiene que:*

$$E_{sel,cru,mut}(\xi(Q,t+1)) \geqslant \xi(Q,t)\cdot(\text{eval}(Q,t)/\overline{F(t)})\left[1 - p_c\frac{\delta(Q)}{l-1} - \mathcal{O}(Q)p_m\right] \tag{2.9.1}$$

Demostración. La demostración se efectuará en tres pasos, en el primero se estudiará el efecto de la selección basada en la ruleta sesgada, sobre el número esperado de individuos que se asocian con la ristra Q. Dicho número se denotará por $E_{sel}(\xi(Q,t+1))$. A continuación se verá el efecto del cruce en dicho número esperado, el cual se denotará por $E_{sel,cru}(\xi(Q,t+1))$, para finalmente tratar el efecto de la mutación, obteniéndose una cota inferior para $E_{sel,cru,mut}(\xi(Q,t+1))$.

Efecto Selección

Utilizando la notación anteriormente introducida, se tiene que la probabilidad de que un individuo seleccionado para cruzarse (suponemos que la selección se efectúa de manera proporcional a la función objetivo) se empareje con el esquema Q, se calculará por medio de $\text{eval}(Q,t)/F(t)$.

El número esperado de individuos que a partir del anterior individuo seleccionado van a emparejarse con Q, será $(\text{eval}(Q,t)/F(t))\cdot\xi(Q,t)$. Al efectuarse λ selecciones, se tendrá que:

$$E_{sel}(\xi(Q,t+1)) = \lambda\cdot\xi(Q,t)\cdot(\text{eval}(Q,t)/F(t)),$$

teniendo en cuenta que $\overline{F(t)} = F(t)/\lambda$, tendremos:

$$E_{sel}(\xi(Q,t+1)) = \xi(Q,t)\cdot(\text{eval}(Q,t)/\overline{F(t)}).$$

Si se asume que la evaluación del esquema supera a la evaluación media extendida a toda la población en un $\varepsilon\,\%$, es decir si

$$\text{eval}(Q,t) = \overline{F(t)} + \varepsilon\overline{F(t)}, \quad (\varepsilon > 0),$$

entonces obtendremos que

$$E_{sel}(\xi(Q,t)) = \xi(Q,0)\cdot(1+\varepsilon)^t. \tag{2.9.2}$$

Por tanto acabamos de obtener que el número medio esperado de individuos que se asocian con un esquema determinado Q en el tiempo t, crece de forma exponencial en función de la diferencia entre la evaluación media de dicho esquema con respecto a la evaluación media global. La expresión (2.9.2) se conoce bajo el nombre de ***ecuación de crecimiento reproductivo de los esquemas***.

Efecto Cruce

Denotando por $p_{d,cruce}(Q)$, la probabilida de que el esquema Q sea destruído por el operador de cruce, se tiene que

$$p_{d,cruce}(Q) = \frac{\delta(Q)}{l-1}.$$

En realidad, como el operador de cruce se lleva a cabo con una cierta probabilidad p_c, se tiene que

$$p_{d,cruce}(Q) = \frac{\delta(Q)}{l-1}\cdot p_c,$$

lo cual equivale a decir que la probabilidad de que el esquema Q sobreviva al cruce es

$$p_{so,cruce}(Q) = 1 - p_{d,cruce}(Q) = 1 - \frac{\delta(Q)}{l-1}\cdot p_c. \tag{2.9.3}$$

La fórmula (2.9.3) nos proporciona una cota para la probabilidad de que sobreviva el esquema Q. Esto es debido a que, puede ocurrir que aunque el punto de corte se encuentre entre los símbolos $*$ comprendidos entre el primer y el último elemento distintos de $*$, el esquema resultante no muera. Así por ejemplo, si consideramos los dos esquemas:

$$\begin{aligned} S_1 &= 1**0, \\ S_2 &= 1**0, \end{aligned}$$

independientemente del punto de corte, cualquiera de los esquemas resultantes coincide con los anteriores. De ahí que en realidad tenemos que:

$$p_{so,cruce}(Q) \geqslant 1 - \frac{\delta(Q)}{l-1}\cdot p_c. \tag{2.9.4}$$

y de (2.9.4) se obtiene que

$$E_{sel,cru}[\xi(Q,t+1)] \geqslant \xi(Q,t)\cdot \text{eval}(Q,t)/\overline{F(t)}\cdot\left(1 - \frac{\delta(Q)}{l-1}\cdot p_c\right). \tag{2.9.5}$$

Efecto Mutación

Denotando por $p_{so,mut}(Q)$ a la probabilidad de que el esquema Q sobrevida al operador de mutación, se tiene que $p_{so,mut}(Q) = (1-p_m)^{\mathcal{O}(Q)}$. Al ser $p_m \ll 1$, la anterior probabilidad de sobrevivir puede ser aproximada por

$$p_{so,mut} \approx 1 - \mathcal{O}(Q)p_m.$$

El efecto combinado de la selección, el operador de cruce y el operador de mutación nos proporciona el denominado Teorema de los Esquemas. Es decir

$$E_{sel,cru,mut}[\xi(Q,t+1)] \geqslant \xi(Q,t)\cdot \mathrm{eval}(Q,t)/\overline{F(t)}\cdot\left[1-\frac{\delta(Q)}{l-1}p_c-\mathcal{O}(Q)p_m+\frac{\delta(Q)}{l-1}\mathcal{O}(Q)p_c p_m\right], \tag{2.9.6}$$

suprimiendo el último término de (2.9.6) obtenemos precisamente (2.9.1). □

§2.9.2. Paralelismo Implícito

El hecho de que los Algoritmos Genéticos efectúen búsquedas robustas, se debe a que implícitamente muestrean hiperplanos. Es decir cada individuo, representado por medio de una codificación binaria, constituye un vértice del hipercubo que representa el espacio de búsqueda, y es a su vez un miembro de 2^l-1 diferentes hiperplanos, donde l es la longitud de la codificación binaria. Por otra parte existen un total de 3^l-1 hiperplanos particiones sobre el espacio de búsqueda total.

Cada individuo no proporciona gran información si se examina por separado. Es por ello que la búsqueda basada en una población es crítica para los Algoritmos Genéticos. Una población de individuos, proporciona información sobre numerosos hiperplanos, estando los hiperplanos de bajo orden muestrados por numerosos puntos de la población. El hecho de que se muestreen muchos hiperplanos cuando se evalúa una población de ristras, se conoce como el paralelismo implícito o intrínseco de los Algoritmos Genéticos. Dicho paralelismo implícito, significa que muchas competiciones entre hiperplanos se efectúan simultáneamente.

§2.9.3. Teoremas de Convergencia

RUDOLPH (1994) demuestra la no convergencia hacia el óptimo global del algoritmo genético canónico, así como la garantía, de convergencia expresada en términos probabilísticos, del algoritmo genético que mantiene a la mejor solución en la población.

DAVIS y PRINCIPE (1993) extrapolan los fundamentos teóricos del algoritmo «***simulated annealing***» a un modelo de algoritmo genético basado en cadenas de Markov. Se efectúa un estudio de las matrices de transición de estados teniendo en cuenta en primer lugar tan sólo la reproducción, a continuación la reproducción y la mutación y finalmente la reproducción, la mutación y el cruce.

SUZUKI (1993) efectúa un estudio de la convergencia de los algoritmos genéticos por medio de cadenas de Markov. Los algoritmos genéticos estudiados presentan un criterio de reducción elitista modificado, según el cual se genera una población de λ individuos, incluyendo en ella al mejor individuo de la población en la generación anterior, obteniéndose los $\lambda-1$ individuos restantes por medio de las operaciones genéticas normales.

LIEPINS (1992) demuestra la convergencia del algoritmo genético hacia poblaciones que contienen al óptimo, en el caso de algoritmos genéticos sin operador de mutación, pero en los cuales el reemplazamiento de individuos es elitista – el mejor individuo no se pierde nunca – y además se efectúa de tal manera que en cada paso cualquier punto del espacio sea potencialmente alcanzable por medio de la operación de cruce.

CHAKRABORTY y DASTIDAR (1993), presentan un modelo de fiabilidad estocástica de un esquema para el algoritmo genético binario con longitud de representación fija, y obtienen una estimación para el número de generaciones necesarias hasta obtener la convergencia.

EIBEN, AARTS y VAN-HEE (1990) modelan la evolución del algoritmo genético por medio de una cadena de Markov, obteniendo condiciones suficientes para la convergencia en probabilidad del procreso evolutivo hacia el óptimo global.

CAPÍTULO 3

ALGORITMOS EVOLUTIVOS PARA OPTIMIZACIÓN MÚLTIOBJETIVO

En este capítulo se van a aplicar los conceptos estudiados en los capítulos previos al Problema de la Ruta más Corta, estocástico multiobjetivo. Primero se presenta algunos conceptos básicos. Luego se muestran los pasos a seguir para resolver este problema

§3.1. DEFINICIÓN DEL PROBLEMA DE RED

Se considera un grafo dirigido o digrafo $G = (N, A)$, donde $N = \{1, 2, \ldots, m\}$ es el conjunto finito de nodos (vértices o puntos) y $A = \{(i, j), (k, l), \ldots, (s, t)\}$ el conjunto de arcos dirigidos que unen pares de nodos en N. Si (i, j) pertenece al conjunto A se dice que j es un nodo adyacente a i. Una red $R = (N, A, f)$ es un grafo dirigido donde además se cuenta con una función $f : A \to \mathbb{R}$ que asigna a cada arco un valor determinado, que representa la distancia o el costo de viajar de un nodo a otro adyacente.

Los diversos modelos, algoritmos y aplicaciones se han desarrollado para tratar una variedad de panoramas de redes tales como ruta más corta, flujo máximo, y problemas de flujo con costos mínimos.

Las redes son muy útiles para describir muchos problemas. Uno de los modelos introducidos en la literatura es el problema de flujo de costos mínimos que se puede utilizar como punto de partida para representar flujo máximo, el problema de la ruta más corta y modelos de transporte.

Para esto consideremos una red que tiene m nodos y n arcos. Cada nodo i se asocia a un número b_i, representante de la oferta de un artículo si es positivo o la demanda si es negativo. Cada arco (i, j) tiene asociado x_{ij} , que es la cantidad de flujo sobre el arco y c_{ij} que es el costo unitario de embarque a lo largo del arco.

El problema de flujo de costo mínimo es embarcar la oferta disponible a través de la red a fin de satisfacer la demanda a un costo mínimo. La representación matemática siguiente es utilizada como un acercamiento de programación lineal para describir este proceso de optimización

(BAZARAA y otros, 1999).

$$\begin{aligned} \text{Minimizar} \quad & \sum_{i=1}^{m}\sum_{j=1}^{m} c_{ij}x_{ij} \\ \text{sujeto a} \quad & \sum_{j=1}^{m} x_{ij} - \sum_{k=1}^{m} x_{ki} = b(i) \quad i = 1,\ldots,m \\ & x_{ij} \geqslant 0 \quad i,j, = 1,\ldots,m \end{aligned} \tag{3.1.1}$$

Este modelo es muy útil para identificar todas las variables implicadas en el proceso de la optimización. El modelo representa un sistema de las ecuaciones que tienen como un objetivo la minimización del costo total incurrido al transportar artículos a partir de un nodo a otro. También demuestra las restricciones que se aplican a las diversas variables. En el modelo, x_{ij} representa el uso de un arco particular en la trayectoria. En este caso x_{ij} tiene limitaciones en la capacidad. En la mayoría de los casos, los límites más bajos en flujos del arco son cero (AHUJA y otros, 1993). El costo de mover los artículos para la mayoría de los modelos determinísticos es una constante pero en este caso podemos cambiar esa constante por funciones con diversas distribuciones discretas o continuas. La definición básica de la red no cambia pero la información entre nodos sí.

§3.2. El Problema de la Ruta Corta

De forma general, en el problema de ruta corta se tiene una red, R que tiene m nodos que representan ciudades, con n arcos o caminos de ciudad a ciudad y los costos c_{ij} asociados con cada arco (i,j), tratándose de encontrar la trayectoria más corta (o la menos costosa) que vaya de un origen a un destino deseado (que puede ser el nodo m). El costo total de la ruta es la suma de los costos de los arcos de la ruta, y las $x_{ij} = 0,1$ indican que cada arco está o no en la ruta. El planteamiento matemático es el siguiente:

$$\begin{aligned} \text{Minimizar} \quad & \sum_{i=1}^{m}\sum_{j=1}^{m} c_{ij}x_{ij} \\ \text{sujeto a} \quad & \sum_{j=1}^{m} x_{ij} - \sum_{k=1}^{m} x_{ki} = \begin{cases} 1 & \text{si } i = 1 \\ 0 & \text{si } i \neq 1 \wedge i \neq m \\ -1 & \text{si } i = m \end{cases} \\ & x_{ij} \geqslant 0 \quad i,j, = 1,\ldots,m \end{aligned} \tag{3.2.1}$$

Este problema es uno de los más estudiados en el área de optimización, dado que existen muchos algoritmos que lo resuelven, dadas ciertas condiciones. Por ejemplo, el algoritmo de Dijkstra lo resuelve cuando todos los costos son constantes no negativas con el objetivo de minimizar la distancia (BAZARAA y otros, 1999).

§3.3. CONDICIONES ESTOCÁSTICAS

La mayoría de los algoritmos se enfocan a solucionar estos modelos considerando costos o distancias como datos determinísticos. Sin embargo, los estudios recientes han incorporado funciones y condiciones de probabilidad al viajar por los nodos (DAVIS y LINGRAS, 2003; RUBINSTEIN, 1981; OSYCZKA, 2002). Debido a que el modelo del que se está hablando incorpora datos tanto determinísticos como probabilísticos, y también los diferentes objetivos que se tienen al viajar sobre los arcos, los principios de los algoritmos evolutivos ayudan a encontrar un buen sistema de soluciones para finalmente obtenerlas en un frente de Pareto.

En esta clase de problemas las expresiones matemáticas son muy complicadas, por lo que sus soluciones se han encontrado al trabajar con técnicas evolutivas. Las nuevas técnicas también proporcionan algunas formas de incorporar más restricciones a un problema multiobjetivo (JIN y otros, 2003). Muchas distribuciones estocásticas junto con sus parámetros se pueden incorporar durante el proceso evolutivo de una manera tal que después de muchas generaciones las mejores trayectorias creen un frente de Pareto para que el usuario decida la mejor ruta para seguir.

§3.4. ALGORITMOS EVOLUTIVOS ESTOCÁSTICOS

Basado en claves aleatorias (random keys, RK), idea estudiada por GOLDBERG y otros,2002, el método propuesto fue desarrollado por el Dr. GÓMEZ-SÁNCHEZ basado en una estructura de datos dinámica para la representación evolutiva del algoritmo. La construcción de trayectorias considera objetivos múltiples a través de una representación al azar del esquema de árbol con un muy pequeño aumento en el tiempo de modelado. También elimina los procesos de validación. La incorporación de la variabilidad, de la probabilidad y de la dependencia en modelos de la red puede conducir a modelos matemáticamente insuperables. El uso de algoritmos evolutivos ofrece una buena alternativa porque se basa en la prueba de las soluciones posibles creadas directamente de una ruta creada durante el proceso evolutivo.

§3.5. REDES ESTOCÁSTICAS MULTIOBJETIVO

El procedimiento utiliza los principios evolutivos de los algoritmos para solucionar el problema de ruta corta donde deseamos enviar un móvil de un punto de partida a un punto final, con el costo mínimo, a través de una red bajo condiciones estocásticas. La generación de un conjunto de rutas es válida de acuerdo con la estructura de la red. Una vez que la primera generación esté lista y validada, las trayectorias se evalúan para obtener los mejores elementos. Las trayectorias seleccionadas se pueden utilizar para crear una nueva generación después de aplicar la recombinación y la mutación. En la nueva generación, las trayectorias pueden contener la información de la generación anterior o pueden ser totalmente nuevas, esto depende de los parámetros seleccionados para la recombinación y la mutación. El algoritmo utiliza una estructura dinámica que permite

que el tamaño del cromosoma sea variable en vez de la longitud fija según lo utilizado en otros trabajos como vector (DAVIS y LINGRAS, 2003; FERNANDEZ y otros, 1998; LI, 2003). Las distribuciones probabilísticas y los pares afectados de nodos se pueden definir en un archivo separado. El sistema de soluciones es afectado por el cruce y la mutación hasta que una condición que detiene el proceso se alcanza para crear un conjunto de Pareto.

§3.6. SIMULACIÓN DE VARIABLES ALEATORIAS

En un modelo, se pueden diferenciar dos tipos de datos: los que no cambian a través del tiempo que se conocen como parámetros y los que presentan cambios a través del tiempo conocidos como variables. La simulación es útil en un modelo en donde la información tiene carácter dinámico y probabilístico, debido a que la interacción de la información es difícil de analizar.

La variabilidad que se presenta en la información que cambia con el tiempo debe modelarse según ciertas ecuaciones matemáticas que sean capaces de reproducirla, que en la mayoría de los casos se puede clasificar dentro de alguna distribución de probabilidad.

Es por esto que para este trabajo se simularon las distribuciones de probabilidad más comunes mediante el uso de un generador de números aleatorios entre 0 y 1 generado por MURATA y otros (2003). A este generador de aleatorios se le hicieron las pruebas de bondad de ajuste χ^2 y la de Kolmogorov-Smirnov según Reuven (ROSS, 2002), para cerciorarse de que los datos de dicho generador se ajustan a una distribución uniforme continua entre 0 y 1, lo que se logró exitosamente.

A partir de dicho generador, se utilizaron métodos de transformaciones para encontrar la distribución de probabilidad de otras variables aleatorias utilizadas comúnmente que se describen a continuación utilizando referencias existentes (KUMAR y BANERJEE, 2003; PRETOLANI, 2000).

§3.6.1. Distribuciones Continuas

Uniforme (a, b)

A partir del generador de aleatorios, podemos obtener valores de una distribución uniforme $(0, 1)$. Al generalizar ésta, podemos obtener valores de una distribución uniforme (a, b) de la siguiente manera:

Sea U una variable aleatoria con distribución uniforme continua $(0, 1)$, si hacemos $X = U(b - a) + a$, entonces $X \sim$ Uniforme continua (a, b).

Exponencial (λ)

Sea U una variable aleatoria con distribución uniforme continua $(0, 1)$, si hacemos $X = (\log(U))/\lambda$, entonces $X \sim$ Exponencial(λ).

Gamma (n, λ)

Sea $E \sim$ Exponencial(λ), entonces si $X = \sum_{k=1}^{n} E, X \sim$ Gamma(n, λ)

Normal $(0, 1)$

Sea $U \sim$ Uniforme continua $(0, 1)$, entonces si $X = (-2 \log U)^{1/2} \cos(2\pi U)$, $X \sim$ Normal$(0, 1)$.

Normal (μ, σ^2)

Sea $U \sim$ Normal$(0, 1)$, entonces si $X = (U\sigma) + \mu$, $X \sim$ Normal(μ, σ^2).

$\chi^2(n)$

Si $X = \sum_{i=1}^{n} Z_i^2$ donde Z_i con $i = 1, \ldots, n$ son normales $(0, 1)$ independientes, entonces $X \sim \chi^2$ con n grados de libertad.

Si $X = Z^2 - 2\log(\prod_{i=1}^{k} U_i)$, donde Z se distribuye normal $(0, 1)$ y U_i con $i = 1, \ldots, n$ son uniformes $(0, 1)$ independientes, entonces X se distribuye χ^2 con $2k + 1$ grados de libertad. A partir de la anterior, una χ^2 con $2k$ grados de libertad se puede simular como $-2\log(\prod_{i=1}^{k} U_i)$, donde U_i con $i = 1, \ldots, n$ son uniformes $(0, 1)$ independientes [34].

§3.6.2. Distribuciones Discretas

Las distribuciones discretas se pueden obtener a partir de su definición, éstas se definen a continuación.

Uniforme Discreta (n)

Sea U una variable aleatoria con distribución uniforme continua $(0, 1)$, si hacemos $X = \lfloor nU \rfloor + 1$, entonces $X \sim$ Uniforme discreta(n).

Binomial (n, p)

Un experimento Binomial consiste en realizar n pruebas donde cada prueba tiene dos posibilidades, éxito o fracaso. La probabilidad de éxito en cada prueba es p. Las pruebas entre sí son independientes. En general la variable aleatoria se define como el número de éxitos de n pruebas.

Bernoulli (p)

La función de distribución Bernoulli es un caso particular de la Binomial, cuando $n = 1$. Es decir, se tiene una probabilidad de éxito p en la única prueba que se realiza.

Poisson (λ)

La distribución de probabilidad del tiempo entre llegadas o bien la tasa de entrada promedio a un sistema, generan un experimento poisson. Las llegadas al sistema se llevan a cabo de forma aleatoria. Por otra parte, una de las propiedades de esta distribución es su relación con el tiempo entre llegadas consecutivas que se representan en forma paralela de acuerdo a un proceso exponencial.

Hipergeométrica (M, K, n)

Un experimento Hipergeométrico consiste en obtener una muestra aleatoria sin reemplazo de tamaño n, de una población de tamaño N, donde ésta se divide en dos subpoblaciones, una de éxitos de tamaño K y otra de fracasos de tamaño $N - K$. La variable aleatoria se define como el número de éxitos en la muestra.

Geométrica (p)

Un experimento Geométrico consiste en realizar varias pruebas idénticas e independientes donde la probabilidad de éxito en cada ensayo es p. La variable aleatoria se define como el número de fracasos antes de alcanzar el primer éxito.

Binomial Negativa (r, p)

Un experimento Binomial Negativo consiste en realizar varias pruebas o ensayos idénticos e independientes. Cada prueba tiene dos opciones: éxito (p) o fracaso $(1 - p = q)$. La variable aleatoria se define como el número de fracasos antes de lograr el r-ésimo éxito.

§3.7. PROGRAMA

Para desarrollar el programa, se utilizaron las herramientas y conceptos anteriormente descritos, para así obtener los resultados de un problema de ruta corta estocástico multiobjetivo.

§3.7.1. Algoritmo que Resuelve el Problema de un Objetivo Determinístico

El programa que se desarrolló está en el lenguaje de programación C++. En el programa se van a tener las funciones de densidad simuladas tal como se describió anteriormente. Se programó inicialmente el problema de optimización de un problema de ruta corta estocástico, siguiendo el algoritmo genético, cuya estructura general se propuso en el capítulo 2, en el Pseudocódigo 2.2.1, cuyas variaciones se muestran a continuación.

Para iniciar, se crea aleatoriamente un conjunto de individuos. Después el algoritmo itera en los siguientes pasos hasta que se cumpla alguna condición. Se evalúa la función $f(x)$ para cada individuo, ordenando la población del mejor individuo al peor. Se selecciona (con alguna regla de

selección) una pareja de individuos a partir de la cual se generan nuevos individuos, que si son aceptados por mejorar la población, se incorporan. Se utiliza la nueva población para correr otra vez el algoritmo. Si la condición está satisfecha, se termina y se devuelve la mejor solución en la población actual. Si no, se evalúa otra vez la función $f(x)$, hasta parar.

§3.7.2. Algoritmo que Resuelve el Problema de un Objetivo Estocástico

El programa que se construyó a partir del algoritmo presentado en la sección anterior se generalizó para que los costos en todos los arcos variaran de acuerdo a funciones de densidad dadas. El único cambio significativo con respecto al anterior es que en el momento de crear las rutas y obtener sus costos, se van recorriendo los arcos de la ruta de uno en uno. Si la ruta tiene q nodos, entonces se obtiene primero el costo de ir del nodo 1 al 2, luego del 2 al 3 y así sucesivamente hasta llegar al nodo q. Al ir pasando sobre los arcos, como cada uno tiene una variable aleatoria asociada, se le llama a la subrutina del programa que obtiene un número que proviene de esa distribución y va a ser el costo asociado para ese arco. Se suman todos los costos obtenidos y tendremos el costo total de la ruta. Algunas aplicaciones las encontramos en HERNÁNDEZ-AGUIRRE y otros (2003); SNYDER y STEELE (1995).

§3.7.3. Algoritmo que Resuelve el Problema Multiobjetivo

Una diferencia importante entre un problema con un objetivo y uno multiobjetivo, es que en el caso multiobjetivo, el concepto de solución óptima es más difícil de definir. No se puede encontrar una única solución al problema, por lo que es necesario aplicar el concepto de óptimos de Pareto. Es por esto que el paso que va a cambiar del algoritmo que se presentó al principio es la selección de los mejores elementos de la población, que van a formar parte del frente de Pareto, selección que está basada en el concepto de óptimo de Pareto.

Es importante destacar que cuando se tiene una nueva solución, ésta puede caer dentro de tres categorías, que se deben de tomar en cuenta en el algoritmo, y son:

- Si la nueva solución domina a por lo menos una de las que se han encontrado hasta ese momento, entonces se trata de un nuevo óptimo de Pareto. Las soluciones dominadas se eliminan del conjunto y la nueva solución se agrega.
- Si la nueva solución no domina a ninguna de las soluciones existentes, pero tampoco es dominada, se trata de un nuevo óptimo de Pareto, el cual se agrega al conjunto.
- No es una nueva solución de Pareto, porque es dominada por alguna de las existentes, por lo que no se realiza ningún cambio en el conjunto de soluciones de Pareto.

Para seleccionar soluciones de Pareto, se toma la primera solución de la población x_1 y se considera como un óptimo de Pareto que se va a denotar por $f^{*1} = f(x_1)$. Se utilizan contadores $R = 1$ y $j = 1$. Si $j = j + 1 \leqslant J$ es cierto, donde J es el número de soluciones en la población,

entonces se toma la solución x_j y se continúa. En otro caso, se detiene, ya que todas las soluciones han sido consideradas. Sea $r = 1$ y $q = 0$, donde q es el número de soluciones a eliminar del frente de Pareto existente. Si para toda $i = 1, 2, \ldots, I$ se cumple que $f_i(x_j) < f_i(x^{*r})$ entonces $q = q + 1$, $f^{*R} = f(x_i)$, recordando la solución que debe ser eliminada y se pasa a la asignación de $r = r + 1$. Si no, se continúa con los pasos. Si para toda $i = 1, 2, \ldots, I$ se cumple que $f_i(x_j) \geqslant f_i(x^{*r})$ se regresa a verificar si $j = j + 1 \leqslant J$. De no ser así, se continúa. Se asignan los valores de $r = r + 1$ y si $r \leqslant R$, y se regresa a verificar si $f_i(x_j) < f_i(x^{*r})$. En otro caso se continúa. Si $q \neq 0$, se actualiza el conjunto de soluciones de Pareto, borrando las soluciones seleccionadas y agregando la solución x_j como una nueva solución del conjunto de Pareto y se regresa hasta recorrer todas las soluciones. En otro caso, se continúa y se asigna $R = R + 1$, $x^{*R} = x_j$, $f(x^{*R}) = f(x_j)$ y se itera hasta probar todas las soluciones. En la figura3.7.1 se presenta el diagrama del algoritmo a utilizar para seleccionar soluciones de Pareto (NISHIMURA y MATSUMOTO, 1998).

§3.7.4. Algoritmo que Resuelve el Problema Multiobjetivo de Ruta Corta Estocástico

Para resolver el problema multiobjetivo de ruta corta estocástico, se va a trabajar conjuntamente con los dos últimos algoritmos expuestos, el que hace la parte estocástica y el que trabaja con varios objetivos. Al juntar esos dos principios se implementó el programa para la solución del problema.

Como parte del desarrollo para solucionar el problema multiobjetivo de ruta corta estocástico, se llevaron a cabo diferentes procesos que ayudaron a dar una solución al problema. Como principales elementos del problema, se tienen los siguientes, que se van a describir en las siguientes secciones de este capítulo.

- Datos
- Algoritmo
- Resultados

§3.8. DATOS

Dentro del problema multiobjetivo de ruta corta estocástico, se tienen muchos datos de entrada, por lo que el programa que se desarrolló lee de un archivo toda la información necesaria para poder resolver el problema. La primera línea que lee es el nombre del archivo. En la siguiente línea se tiene la descripción del grafo que se va a trabajar, leyéndose el número de nodos (m) que contiene la red y el número de arcos que unen los nodos (n). A partir de la siguiente línea, se lee en cada línea la descripción de cada uno de los arcos que forman el grafo.

Cada arco tiene un nodo origen y un nodo final, que en el archivo están representados por números, es decir el nodo del que se desea partir es el número 1 y el último nodo, al que se desea

llegar, es el m-ésimo. Analizando únicamente un objetivo, como un caso particular del multiobjetivo, el costo en que se incurre al ir de un nodo i a otro adyacente j se puede dividir en dos tipos que son fijo y variable.El costo total que existe de ir del nodo i al j, sería la suma del costo fijo más el costo variable, es decir

$$CT_{ij} = cf_{ij} + cv_{ij} \tag{3.8.1}$$

donde

CT_{ij} es el costo total de ir del nodo i al j

cf_{ij} es el costo fijo de ir del nodo i al j

cv_{ij} es el costo variable de ir del nodo i al j

El costo fijo está dado de una forma totalmente determinística, en cambio el costo variable, está dado mediante funciones de distribución que se acerquen a la forma como se comporta dicho costo en la realidad. Para lo anterior, se hicieron simulaciones de las principales variables aleatorias, utilizando un generador de números aleatorios, que mediante transformaciones, se obtienen números que distribuyen de cierta manera. Muchas veces, las distribuciones tienen diferentes parámetros, por ejemplo los parámetros de una distribución normal, son su media y varianza. Por lo descrito anteriormente, en los datos de entrada se deben de especificar los nodos que forman el arco, el costo fijo, la distribución que tiene el costo variable y sus parámetros correspondientes, en el caso de un solo objetivo. Toda la explicación anterior se puede aplicar al caso multiobjetivo definiendo p como el número de objetivos que se desean optimizar. Cada arco tiene p diferentes costos, donde cada uno describe un objetivo en específico.

Los datos que entran desde el archivo, son aquéllos que describen el arco, como ya se mencionó, los nodos que lo forman y los p costos, de la siguiente forma. El primer número es el nodo inicial i, el segundo es el nodo final j, luego el costo fijo del primer objetivo, una clave que tiene asignada cada distribución que debe estar seguida de los valores de sus parámetros según la Tabla 3.8.1; el costo fijo del segundo objetivo, una clave de la distribución y valores de los parámetros; el costo fijo ... y así hasta el p-ésimo objetivo.

La representación de la entrada de los arcos es como se muestra en la Tabla 3.8.2. Todos los datos que se leen del archivo de entrada son guardados en diferentes estructuras que ayudan a que el programa se ejecute en un tiempo razonable, utilizando algunas que no se habían usado para resolver el problema, que ayuda a mejorar el tiempo, que se van a describir en secciones subsecuentes.

§3.9. ALGORITMO

El Pseudocódigo 3.9.1 describe el algoritmo evolutivo de la trayectoria más corta estocástica multiobjetivo desarrollado para solucionar el problema de ruta corta estocástica multiobjetivo bajo funciones probabilísticas discretas o continuas. Fue desarrollado usando código estándar de C++ así que puede ser portable a muchos sistemas operativos.

TABLA 3.8.1: Claves asignadas a las Distribuciones

Clave	Distribución	Parámetros
a	Exponencial	λ
b	Gama	n, λ
c	Normal(0,1)	
d	Normal	μ, σ^2
e	Uniforme Continua(0,1)	
f	Uniforme Continua	a, b
g	χ^2	n-grados de libertad
h	Poisson	λ
i	Hipergeométrica	M, K, n
j	Geométrica	p
k	Binomial	n, p
l	Uniforme Discreta	n
m	Bernoulli	p
n	Binomial Negativa	r, p

TABLA 3.8.2: Representación de la Entrada

Nodos		Costo 1			.....	Costo p		
		Fijo	Estocástico			Fijo	Estocástico	
1	i	$cf_{1,1,i}$	clave	parámetros	.....	$cf_{p,1,i}$	clave	parámetros
i	j	$cf_{1,i,j}$	clave	parámetros	.....	$cf_{p,i,j}$	clave	parámetros
...					.....			
k	m	$cf_{1,k,m}$	clave	parámetros	.....	$cf_{p,k,m}$	clave	parámetros

§3.10. ESTRUCTURAS QUE AYUDAN A LOS ALGORITMOS GENÉTICOS

Las estructuras de datos son uno de los aspectos más importantes del desarrollo de nuevos algoritmos. La manera de codificar una trayectoria tiene una importancia especial para desarrollar un algoritmo genético porque define como entran y se modifican los datos. Si el procedimiento tiene una generación al azar, el paso lógico es crear procedimientos de validación. Las estructuras de datos se utilizan para almacenar la definición y ocasionalmente los parámetros de un problema particular. Es a veces bastante flexible leer los parámetros y los datos, haciéndose más sencillo para el programa y para el usuario. Las estructuras de datos fueron diseñadas con dos propósitos: simplificación de la búsqueda y minimización de los procedimientos de evaluación.

§3.10.1. Rutas Aleatorias y Estructuras Dinámicas

El problema de la ruta corta es particularmente difícil porque una trayectoria puede contener un número variable de nodos. Además, las técnicas al azar de la generación tales como algoritmos genéticos pueden conducir a que los cromosomas representen trayectorias inválidas. Las estructuras de datos basadas en las trayectorias al azar tomadas de árboles al azar es una manera muy

PSEUDOCÓDIGO 3.9.1: Pseudocódigo del Algoritmo Evolutivo para la Ruta más Corta Estocástica Multiobjetivo

```
Comenzar
    Leer red e información estocástica;
    Crear n-rutas;
    Mientras no se cumpla la condición de parada: hacer
        Comenzar
            Actualizar Frente de Pareto;
            Aplicar mutación;
            Aplicar recombinación;
            Crear rutas nuevas;
        Terminar
        Imprimir Resultados;
Terminar
```

simple de crear a una población inicial sin tener que aplicar procedimientos de la validación en los pasos futuros. Los cromosomas en este caso son dinámicos en el sentido de que pueden varias de tamaño, lo que proporciona una manera más rápida de evaluar una población para seleccionar los mejores candidatos.

§3.10.2. Estructura Dinámica de Cromosomas

Para almacenar la información contenida en una ruta, la nueva aplicación utiliza una celda para cada nodo contenido en ella. Si la ruta i tiene n_i nodos, el número de celdas usadas por cromosoma es n_i. Considerando una población de tamaño p, el número total de celdas requeridas es

$$C_t = \sum_{i=1}^{p} n_i \tag{3.10.1}$$

donde C_t es el total de número de celdas usadas en el programa.

§3.10.3. Lista Adyacente

El uso de la lista de adyacencia juega un papel dominante en ambientes estocásticos porque contienen toda la información necesaria para incluir en una trayectoria mientras ésta se genera en el proceso al azar. Las estructuras dinámicas contienen la información sobre el grafo incluyendo nodos y arcos. La lista de adyacencia contiene un sistema de nodos conectados con otro nodo. Esta lista permite conocer las maneras disponibles de continuar una trayectoria de un nodo específico. La lista de adyacencia es muy importante al construir una trayectoria nueva porque podemos crear directamente una trayectoria válida en vez de las trayectorias totalmente al azar así que no es necesario utilizar procedimientos adicionales para la validación. Del nodo que comienza, todos los nodos tienen que tener una lista de adyacencia, a excepción del nodo nulo, que es pasado para todas las trayectorias válidas. Una restricción en la definición del gráfico es que cada nodo tiene

que ser conectado con por lo menos un nodo. Un grafo dirigido garantiza que el nodo siguiente sea parte de una trayectoria. La estructura dinámica presentada requiere una cantidad variable de memoria dependiendo del outdegree de cada nodo. Si O es el numero del outdegree (grado de salida), la memoria total requerida es

$$M = (n-1) + \sum_{i=1}^{n-1} O_i \tag{3.10.2}$$

donde M es el número total requerido por celda, n es el número de nodos y O_i es el número de outdegree para el nodo i.

§3.11. Búsqueda de Soluciones en un Espacio Estocástico Utilizando EA

Se construye el espacio de búsqueda estocástico cuando un par de nodos tiene una definición probabilística de la función de costo, el tiempo o la distancia de viajar a partir de un nodo a otro. La estructura dinámica ofrece la mejor opción para almacenar la información que viene de un grafo para aplicar algoritmos genéticos. El uso de las listas de acoplamiento para almacenar órdenes ahora se ha ampliado al concepto de contenedores, que se pueden definir como estructuras, una lista o vectores que pueden almacenar los datos. Los datos almacenados en una trayectoria contienen la información sobre el nodo que comienza y la ruta para seguir sin importar que la función sea estocástica o no.

§3.11.1. Población Inicial

La población es un conjunto de soluciones posibles al problema. El número de soluciones depende de las condiciones especificas de la red para aplicar el algoritmo genético. La población con número pequeño de nodos en una trayectoria puede ser más grande que la población con número grande de nodos en una trayectoria porque puede contener solamente algunas trayectorias. El uso de una configuración de red particular depende del problema específico y el número de los nodos incluidos en una trayectoria y se puede cambiar según las restricciones del modelo. Cuando el problema puede generar una gran cantidad de combinaciones a explorar, es posible introducir técnicas de programación dinámica para mejorar su rendimiento. Los algoritmos genéticos no tienen que probar todas las combinaciones posibles para obtener un resultado razonable en cierta cantidad de tiempo. Aunque el método de selección alinea y corta a individuos, podría ser cambiado si la estructura de la red lo requiere. Para fijar un número apropiado de generaciones tenemos que considerar el siguiente:

- Número de nodos
- Número de rutas

- Tamaño de las posibles rutas

Analizando muchas configuraciones de red en la literatura (AHUJA y otros, 1993), podemos reconocer que dos casos extremos pueden ocurrir. El primer caso ocurre cuando tenemos una red esparza. Esto significa un número pequeño de conexiones y una gran cantidad de nodos. El segundo caso es cuando la red es muy densa. Esto significa que el número de conexiones es más grande que el número de nodos. Una de las ventajas de usar las estructuras dinámicas es que para las redes grandes no tenemos que crear poblaciones grandes (FERNANDEZ y otros, 1998).

§3.11.2. Encontrando un Frente de Pareto

Debido a muchos cambios en el ambiente, las redes con condiciones estocásticas pueden tener más de una trayectoria que resulta con el mismo conjunto de nodos pero diversos costos o pueden tener un diverso conjunto de nodos con el mismo costo durante el proceso de generar soluciones a través de las generaciones. Todo depende del tipo de funciones estocásticas usadas entre los nodos. El algoritmo evalúa cada un objetivo en muchas trayectorias en vista de las funciones probabilísticas definidas al principio. Alineando y cortando, guarda los mejores candidatos cada vez para compararlos con respecto a los nuevos individuos para cada objetivo de una manera tal que guarden a los mejores individuos para crear el frente de Pareto.

§3.12. RESULTADOS

El programa está hecho de tal forma que se puede ejecutar desde una computadora personal (PC). Al correr el programa se obtienen dos archivos de salida que nos facilitan la comparación de los datos obtenidos en los frentes de Pareto. Uno de los archivos nos sirve para poder graficar los frentes de Pareto. Si el programa se corriera n veces, otra salida nos permite crear el gráfico que integra los n frentes de Pareto para poder ver la región en donde posiblemente se encuentra la «*solución óptima*» del problema. Y una segunda salida nos permite trabajar los datos como tablas donde podemos ver las rutas obtenidas en el frente con sus costos correspondientes que también se pueden analizar. En el siguiente capítulo se resuelve un problema específico aplicando el programa creado. Se muestran los resultados obtenidos y algunas observaciones hechas a estos últimos.

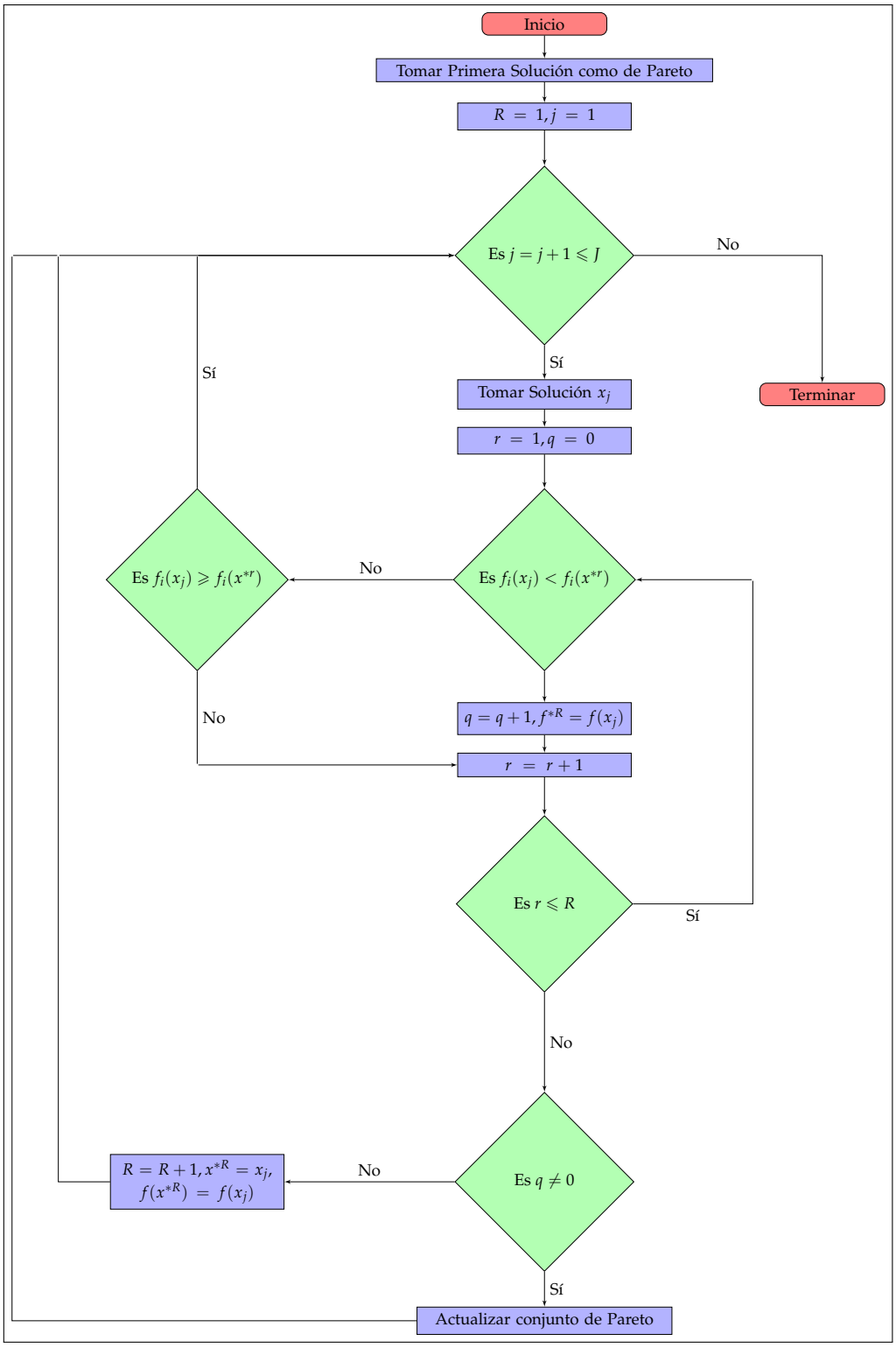


FIGURA 3.7.1: Diagrama de flujo para encontrar soluciones de Pareto

CAPÍTULO 4
RESULTADOS

§4.1. INTRODUCCIÓN

Esta sección presenta los resultados obtenidos del uso del algoritmo genético aplicado a un problema de ruta corta estocástico multiobjetivo. El objetivo es reducir al mínimo dos objetivos (costo y tiempo) de viajar de un nodo inicial A a uno final P en una red con 16 nodos. Se comienza con la definición de la red que muestra la estructura básica del grafo y de datos. La definición muestra la red donde todos los arcos tienen datos determinísticos. Entonces una segunda red muestra la incorporación de la variabilidad y la probabilidad para algunos de los arcos y de los cambios necesarios que tienen que ser incorporados en el algoritmo. Un análisis de las redes seleccionadas, de los parámetros y de los resultados también revela las consideraciones importantes para esta técnica de búsqueda. La red seleccionada es más grande que una red típica, según los trabajos enumerados en las referencias.

Los costos asociados para viajar entre cada par de nodos son determinísticos para el primer caso y estocásticos en el segundo caso. En la Figura 4.2.1 se muestra el diagrama correspondiente de esta red con costos determinísticos.

§4.2. RESULTADOS DETERMINÍSTICOS

El primer experimento para probar el nuevo algoritmo fue conducido en una red determinística con 16 nodos demostrados en la figura 4.2.1. Cada uno de estos arcos tiene dos costos asociados (costo y tiempo) de viajar a partir de un nodo a otro. Al funcionar el programa, el algoritmo genera una población inicial mostrada en la Tabla 4.2.1. Esta población inicial es una generación al azar de los cromosomas que representan las trayectorias. La tabla también muestra el costo asociado para cada trayectoria y el número de los nodos implicados en las trayectorias. Una característica especial de este problema es que una trayectoria de la solución puede contener un diverso número de nodos. La nueva estructura dinámica ofrece ventajas para almacenar esta información en los cromosomas y el costo es parte de esta estructura.

El proceso de la evolución comienza con la mutación, quitando un porcentaje de las trayectorias. Si el porcentaje de la mutación es el 30 % en una población de diez, entonces tres cromosomas son totalmente nuevos. Y si hay un porcentaje de recombinación del 30 %, el 30 % de los nodos de los cromosomas restantes se desechan. Con un porcentaje de recombinación del 30 %, el 70 % de

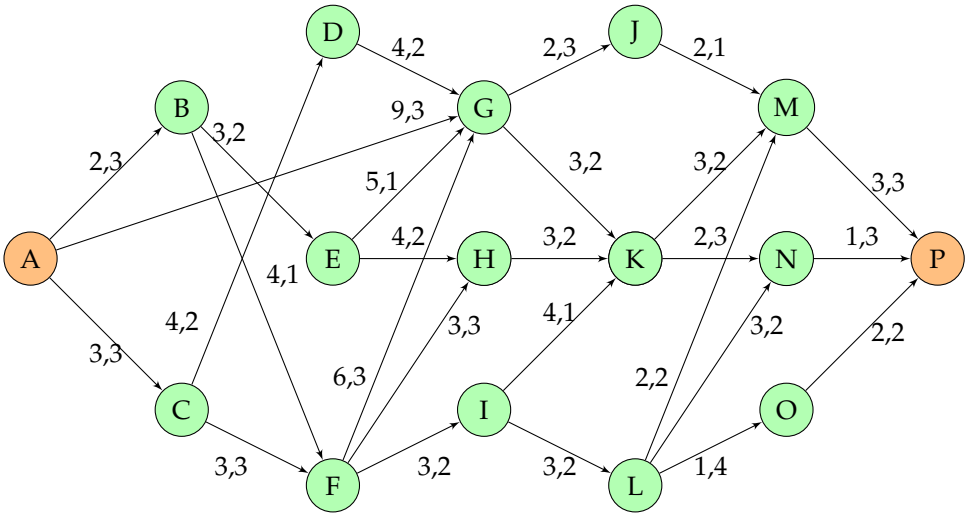


FIGURA 4.2.1: Red determinística con 16 nodos

TABLA 4.2.1: Población inicial para el problema determinístico

Rutas	Tiempo (horas)	Costo (BsF)
(A, B, F, I, K, M, P)	19	12
(A, G, K, M, P)	18	10
(A, G, J, M, P)	16	10
(A, C, D, G, K, N, P)	17	15
(A, C, F, I, L, O, P)	15	16
(A, C, F, I, K, M, P)	19	14
(A, C, D, G, K, M, P)	20	14
(A, C, D, G, J, M, P)	18	14
(A, G, K, N, P)	15	11
(A, C, F, H, K, N, P)	15	17

la información contenida en una trayectoria son transmitidos a la generación siguiente para crear una trayectoria nueva posible y para competir con los nuevos individuos. El paso siguiente es la regeneración de los cromosomas para terminar las trayectorias hasta el nodo blanco. Este proceso se repite para cada generación. El número de las generaciones para alcanzar una solución razonable depende del problema particular. El porcentaje de mutación del 30recombinación del 30 % como los parámetros seleccionados junto con la combinación de diverso número de generaciones dieron lugar a una buena discriminación y evolución del conjunto de las soluciones para este problema particular. Según los resultados, encontrar el conjunto de Pareto, el algoritmo requiere menos de 50 generaciones (50G) y menos de un segundo de tiempo. El experimento también fue conducido con 100 y 1000 generaciones (100G y 1000G) sin cambios significativos en los resultados y aún con menos de un segundo de tiempo. El experimento también fue realizado con porcentajes más altos (porcentaje de recombinación del 50 % y de mutación del 50 %) para considerar los efectos sobre la

TABLA 4.2.2: Frente de Pareto del caso determinístico

Rutas	Tiempo (horas)	Costo (BsF)
(A, G, J, M, P)	16	10
(A, G, K, N, P)	15	11

solución y la mejor solución también fue obtenida con menos de 50 generaciones. Si analizamos la combinación de soluciones posibles, está claro que más cambios en los cromosomas y las trayectorias conducen a diversas soluciones y a menor enfoque en los mejores individuos. El resultado del uso del programa al problema determinístico multiobjetivo se muestra en la Tabla **??** donde se representa el conjunto de Pareto para este problema particular.

§4.3. RESULTADOS DE LA RED ESTOCÁSTICA

Cuando un par de arcos contiene datos probabilísticos, una red se llama red estocástica. La red estocástica representa un desafío más grande para las formulaciones matemáticas porque aumenta el número y el tipo de variables. Al usar el nuevo algoritmo evolutivo desarrollado, la formulación del sistema principal no cambia. Utiliza el mismo algoritmo con solamente una diferencia: funciones de la variabilidad. La incorporación de los cambios, variables y/o probabilidad, es relativamente simple con el nuevo procedimiento propuesto para obtener la función objetivo. La función objetivo contiene una llamada a una rutina adicional para que agregue el costo necesario a cualquier par de nodos. El costo puede ser una variable del tiempo, distancia, carga, o cualquier otra medida y pueden contener una probabilidad asociada a la función. En este caso, cada costo tiene una distribución normal con media (c_{ij}) como en el caso determinístico y una varianza de 1. El archivo de entrada se muestra en el Apéndice A.

$$s_{c_{ij}} = c_{ij} + X; \text{donde } X \sim \text{Normal}(0, 1)$$

La prueba del nuevo algoritmo para una red con los elementos estocásticos fue realizada usando porcentaje de recombinación del 50 % y de mutación del 50 %. La Figura 4.3.1 representa el gráfico del frente de Pareto y la Tabla 4.3.1 corresponde a los resultados (conjunto de Pareto) para 100 generaciones.

La solución de la Tabla 4.3.2 cuyo frente graficado es la Figura 4.3.2 fue obtenida de otra corrida del programa. Como se puede notar, pueden resultar frentes muy diferentes a pesar de que se trata del mismo problema, ya que la parte estocástica hace que varíe de una a otra corrida.

Analizando las rutas que salieron en ambos conjuntos de Pareto, podemos ver que hay rutas que aparecen constantemente, se podría decir que esas rutas pueden ser de las mejores. En el primer frente se puede ver que la ruta A, G, K, N, P salió 3 veces de las 7 soluciones obtenidas. En el segundo caso, en cambio no se puede decir con exactitud cual sería la mejor ruta.

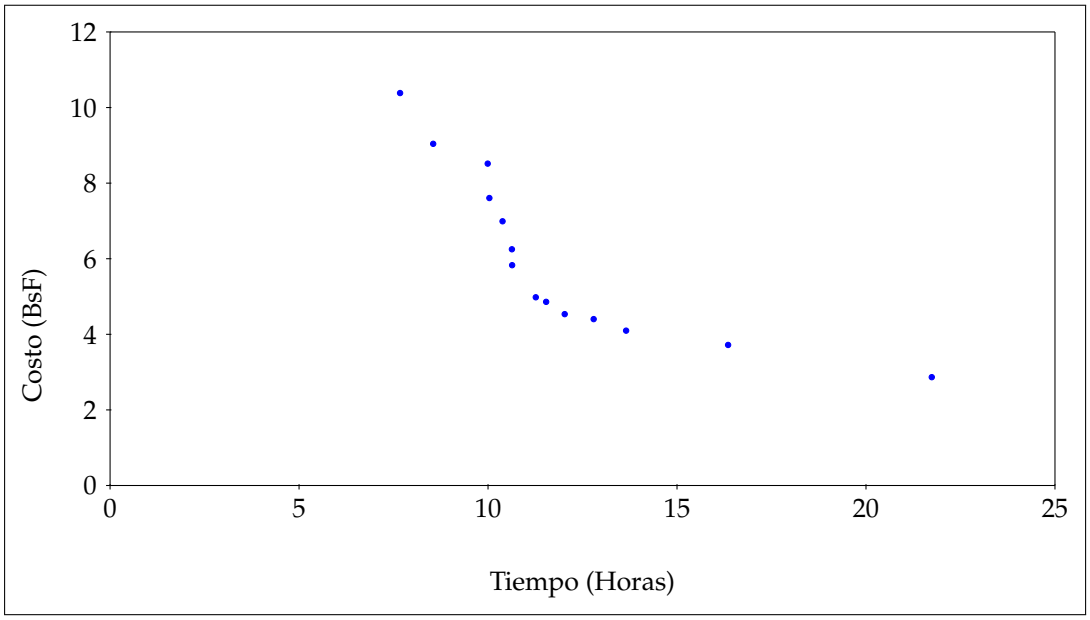


FIGURA 4.3.1: Frente de Pareto para una red estocástica de 16 nodos y 27 arcos utilizando distribuciones Normales entre los arcos (Corrida I)

TABLA 4.3.1: Conjunto de soluciones caso estocástico (100G, 50 % recombinación, 50 % mutación) (Corrida I)

Rutas	Tiempo (horas)	Costo (BsF)
(1, 7, 10, 13, 16)	11.2633	4.97974
(1, 2, 5, 7, 11, 14, 16)	7.67274	10.3836
(1, 7, 11, 14, 16)	10.3838	6.99049
(1, 7, 10, 13, 16)	9.99125	8.51628
(1, 7, 10, 13, 16)	10.6375	5.82899
(1, 7, 10, 13, 16)	12.7946	4.40213
(1, 7, 11, 14, 16)	10.6313	6.24999
(1, 7, 11, 14, 16)	12.0284	4.53388
(1, 3, 4, 7, 10, 13, 16)	21.7403	2.86604
(1, 2, 6, 9, 11, 14, 16)	8.54869	9.0403
(1, 7, 10, 13, 16)	11.5374	4.85899
(1, 7, 10, 13, 16)	16.3516	3.71864
(1, 7, 10, 13, 16)	10.0354	7.60593
(1, 2, 5, 7, 10, 13, 16)	13.6531	4.09724

Se corrió 10 veces el programa obteniéndose 10 diferentes conjuntos de Pareto, que se graficaron conjuntamente, de modo que se visualice el área en que se acumulan más las soluciones, dentro de las cuales podría encontrarse la «***solución óptima***». La gráfica 4.3.3 presenta las soluciones obtenidas de 10 corridas del programa para el problema descrito.

Para hacer un análisis estadístico sobre el comportamiento de los objetivos (tiempo y costo), se utilizaron los 10 frentes de Pareto que están graficados en la Figura 4.3.3. Se determinó la frecuen-

TABLA 4.3.2: Conjunto de soluciones caso estocástico (100G, 50 % recombinación, 50 % mutación) (Corrida II)

Rutas	Tiempo (horas)	Costo (BsF)
(1, 7, 11, 14, 16)	7.58896	7.5532
(1, 7, 10, 13, 16)	9.54309	6.78422
(1, 7, 10, 13, 16)	11.4116	4.70067
(1, 7, 10, 13, 16)	12.9652	2.62531
(1, 7, 10, 13, 16)	6.77845	8.27002

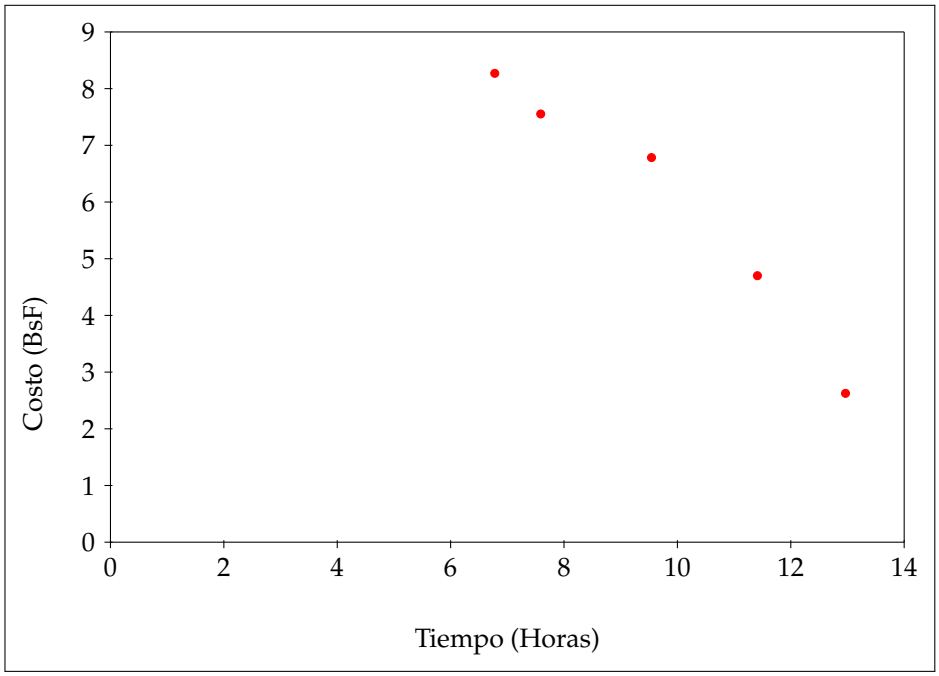


FIGURA 4.3.2: Frente de Pareto para una red estocástica de 16 nodos y 27 arcos utilizando distribuciones Normales entre los arcos (Corrida II)

cia con que aparecieron en los 10 frentes de Pareto. Para cada ruta que se tuvo, se le sacó su media y desviación estándar, tanto del tiempo como del costo, valores que se muestran en la Tabla 4.3.3. Dicha tabla se puede presentar a la persona que toma las decisiones, de tal forma que sirve como información extra para poder decidir. Se puede analizar de tal forma que las mejores rutas, son aquéllas que tienen menor desviación estándar.

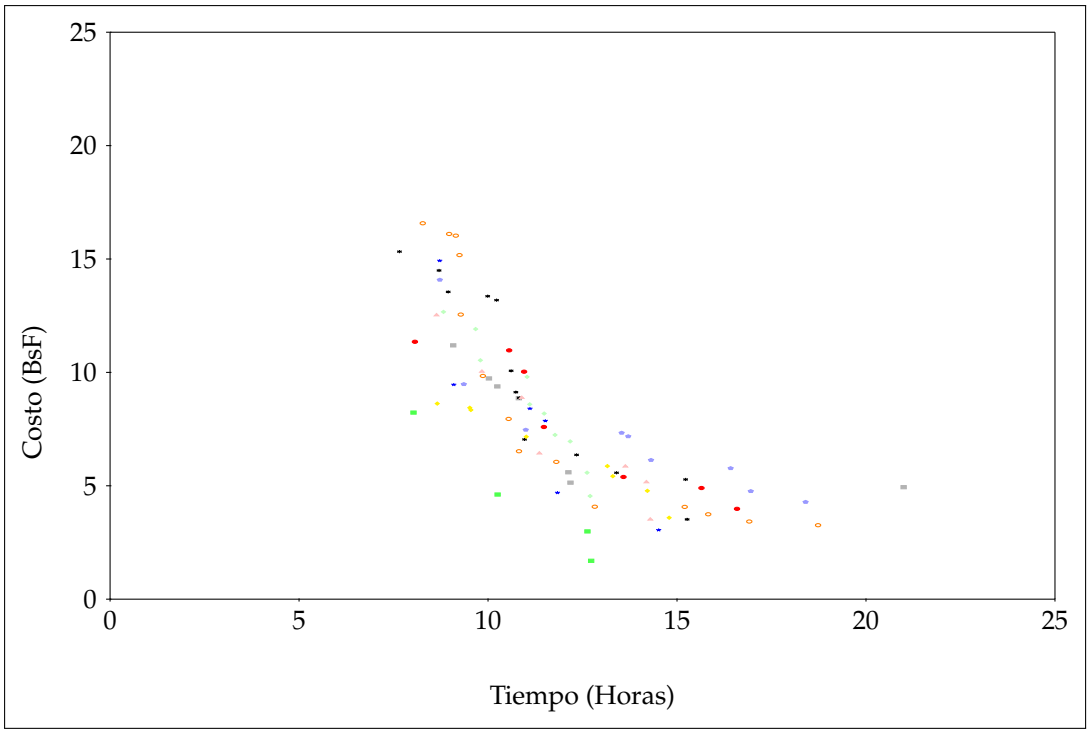


FIGURA 4.3.3: Frentes de Pareto con funciones normales (0, 1) en los arcos.

TABLA 4.3.3: Medias y desviaciones estándar obtenidas a partir de los 10 frentes de Pareto

Rutas	Frec.	Promedio 1	Promedio 2	D.E. 1	D.E. 2
(1, 7, 10, 13, 16)	55	12.1929	5.1553	2.8692	1.9619
(1, 7, 11, 14, 16)	25	10.5369	6.3525	1.5890	1.9852
(1, 2, 6, 9, 12, 15, 16)	8	7.9068	11.5067	1.0146	3.3555
(1, 2, 6, 8, 11, 14, 16)	7	6.8963	14.1985	0.9420	2.7075
(1, 7, 11, 13, 16)	7	15.9551	3.0180	2.4635	0.9560
(1, 3, 6, 8, 11, 14, 16)	5	6.9372	15.2836	1.0083	3.1066
(1, 2, 6, 9, 11, 14, 16)	4	8.7018	9.0743	0.9985	2.6335
(1, 2, 5, 7, 10, 13, 16)	4	12.8529	6.8126	6.2069	4.2011
(1, 2, 5, 8, 11, 14, 16)	3	6.4665	13.9127	0.8728	2.7388
(1, 2, 5, 7, 11, 14, 16)	3	9.2600	9.2150	2.1934	3.9115
(1, 3, 6, 9, 12, 14, 16)	2	6.3907	14.3096	0.7893	0.3131
(1, 2, 6, 9, 12, 14, 16)	1	7.5175	10.4429		
(1, 3, 4, 7, 10, 13, 16)	1	21.7403	2.8660		

CONCLUSIONES

Se ha desarrollado un algoritmo evolutivo para solucionar el problema de encontrar un conjunto de las trayectorias más cortas en una red donde las condiciones estocásticas y multiobjetivo pueden ser incorporadas. Los costos asociados a los arcos se pueden incorporar en la definición de la red para cada par de nodos sin tener que modificar el modelo básico. Se aprovechó de las trayectorias aleatoriamente generadas para crear y evaluar una población muy dinámica a través de muchas generaciones. También se presentan algunos resultados determinísticos y estocásticos para crear un frente de Pareto. Aunque se utilizan solamente dos objetivos, el algoritmo se puede modificar fácilmente para incluir muchos objetivos a la misma red. Como se pudo ver en el desarrollo de este documento, este tipo de problemas es muy difícil de resolver debido a la complejidad que tienen por la parte estocástica.

Sin embargo los algoritmos evolutivos son una alternativa eficiente que proporciona soluciones buenas en tiempos muy cortos. Como se expuso en el capítulo 4, en los resultados, el tiempo en el que corre el programa depende de factores como el número de nodos, la densidad de aristas en el grafo, así como la complejidad de las funciones estocásticas que componen la red. En general el programa es eficiente porque puede resolver problemas con miles de nodos, lo cual se aproxima a problemas reales, en no más de 45 segundos.

Conviene resaltar que un método tradicional (programación dinámica o algoritmos de exploración) no se puede aplicar de manera directa ya que los datos con los que se cuenta se comportan de manera estocástica. Por su naturaleza, la programación evolutiva es una buena herramienta para resolver problemas de este tipo, además de que reduce considerablemente el número de operaciones necesarias para explorar todas las soluciones factibles.

Es muy importante señalar también que las soluciones obtenidas no son totalmente las «***soluciones óptimas***», sino que puede ser la forma como se comportan las mejores soluciones del problema. El programa genera los frentes de Pareto que graficados pueden ayudar a visualizar el comportamiento de las soluciones obtenidas. Con un buen conocimiento del problema que se está tratando de resolver, la persona que toma la decisión puede optar por una, o tal vez hacer una combinación de varias soluciones.

Ya que no existen referencias en las que se trata conjuntamente el problema multiobjetivo con valores estocásticos, se trabajó con los conceptos publicados. Es por esto que se juntaron los conceptos existentes como intento para resolver el problema. Ahora que se terminó, se ha visto que es necesario introducir más conceptos estadísticos para aplicarlos en el caso de más de un objetivo.

Por ejemplo, los resultados encontrados pueden ser analizados estadísticamente para poder tomar decisiones en base a más información que se obtenga de las diferentes corridas del programa.

Finalmente, el área a la que pertenece esta investigación es relativamente nueva, por lo que los conceptos y resultados son aún muy limitados. Por ejemplo, no se ha desarrollado una definición de óptimos de Pareto en el caso estocástico. Se propusieron ideas para atacar el problema y resolverlo de manera eficiente. A partir de este inicio, es necesario que se desarrollen los conceptos aún más aplicados a problemas estocásticos multiobjetivos. Por otra parte, la aplicación actual simula únicamente variables independientes, pero en situaciones reales los objetivos pueden estar relacionados entre sí. Sería importante poder simular objetivos relacionados para tratar ese caso.

REFERENCIAS

AHUJA, R. K.; MAGNANTI, T. L. y ORLIN, J. B. (1993). *Network flows: Theory, Algorithms and Applications*. Prentice Hall.

AVERBAKH, I. y BERMAN, O. (1998). «Location problems with grouped structure of demand: complexity and algorithms». *Networks*, **Vol. 31**, pp. 81–92.

BAZARAA, MOKHTAR S.; JARVIS, JOHN J. y SHERALI, HANIF D. (1990). *Linear programming and network flows*. Wiley, 2ª edición.

BAZARAA, MOKHTAR S.; JARVIS, JOHN J. y SHERALI, HANIF D. (1999). *Programación Lineal y Flujo en Redes*. Limusa.

BROWN, MARTIN y SMITH, ROBERT E. (2003). «Effective use of directional information in multi-objective evolutionary computation». *GECCO*, pp. 778–789.

CHAKRABORTY, U. K. y DASTIDAR, D. G. (1993). «Using reliability analysis to estimate the number of generations to convergence in genetic algorithms». *Information Processing Letters*, **Vol. 43**, pp. 199–209.

CHEN, Y. L.; RINKS, D. y TANG, K. (2001). «The first k minimum cost paths in a time-schedule network». *Journal of the Operational Research Society*, **Vol. 52**, pp. 102–108.

CRUZ-CORTÉS, NARELI y COELLO, CARLOS A. (2003). «Multiobjective optimization using ideas from the clonal selection principle». *GECCO*, pp. 158–170.

DARWIN, CHARLES (1859). *On the Origin of Species by Means of Natural Selection*. Murray.

DAVIS, CEDRIC y LINGRAS, PAWAN (2003). «Genetic algorithms for rerouting shortest paths in dynamic and stochastic networks». *European Journal of Operational Research*, **Vol. 144**, pp. 27–38.

DAVIS, T. E. y PRINCIPE, J. C. (1993). «A Markov chain framework for the simple genetic algorithm». *Evolutionary Computation*, **Vol. 1(No. 3)**, pp. 269–288.

DE-JONG, K. A. (1975). *An analysis of the behaviour of a class of genetic adaptive systems*. Tesis doctoral, University of Michigan.

EIBEN, A. E.; AARTS, E. H. L. y VAN-HEE, K. M. (1990). «Global convergence of genetic algorithms: An infinite Markov chain analysis». *Computing Science Notes*, **(Eindhoven University of Technology, The Netherlands)**.

FERNANDEZ, A. A.; ARMACOST, R. L. y PET-EDWARDS, J. J. (1998). «Understanding simulation solutions to restore constrained project scheduling problems with stochastic task duration». *Engineering Management Journal*, **Vol. 4(Nro. 10)**, pp. 5–13.

FONSECA, C. M. y FLEMING, P. J. (1993). «Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization». En: S. Forrest (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms*, pp. 416–423. Morgan Kauffman Publishers, San Mateo, California.

GÓMEZ-SÁNCHEZ, MIGUEL A. (2001). *Genetic Algorithms and Simulation Applied to Optimization: The Stochastic Shortest Path Model*. Phd dissertation, New Mexico State University, Department of Industrial Engineering.

GOLDBERG, D. E. (1989). *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley.

GOLDBERG, D. E.; ROTHLAUF, F. y HEINZL, A. (2002). «Network random keys - a tree representation scheme for genetic and evolutionary algorithms». *Evolutionary computing*, **Vol. 1(Nro. 10)**, pp. 75–79.

HERNÁNDEZ-AGUIRRE, ARTURO; RIONDA, SALVADOR BOTELLO; COELLO, CARLOS A. COELLO y LIZÁRRAGA, GIOVANNI LIZÁRRAGA (2003). «Use of multiobjective optimization concepts to handle constraints in single-objective optimization». *GECCO*, pp. 573–584.

HOLLAND, JOHN (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor.

HORN, J. y NAFPLIOTIS, N. (1993). «Multiobjective Optimization using the Niched Pareto Genetic Algorithm». *Technical report illigal report 93005*, University of Illinois at Urbana-Champaign, Urbana, Illinois, USA.

ISHIBUCHI, H. y SHIBATA, Y. (2003). «A similarity-based mating scheme for evolutionary multiobjective optimization». *GECCO*, pp. 1065–1076.

JIN, YAOCHU; OKABE, TATSUYA y SENDHOFF, BERNHARD (2003). «Solving three-objective optimization problems using evolutionary dynamic weighted aggregation: Results and analysis». *GECCO*, pp. 636–637.

KUMAR, R. y BANERJEE, N. (2003). «Multicriteria network design using evolutionary algorithm». *GECCO*, pp. 2179–2190.

LAW, AVERILL M. y KELTON, W. DAVID (1982). *Simulation Modeling and Analysis*. McGraw Hill.

LI, XIAODONG (2003). «A non-dominated sorting particle swarm optimizer for multiobjective optimization». *GECCO*, pp. 37–48.

LIEPINS, G. E. (1992). «On global convergence of genetic algorithms». *Neural and Stochastic Methods in Image and Signal Processing*, pp. 61–65.

MICHALEWICZ, Z. (1992). *Genetic Algorithms + Data Structures = Evolution Programs*. Berlin Heidelberg.

MURATA, T.; KAIGE, S. y ISHIBUCHI, H. (2003). «Generalization of dominance relation-based replacement rules for memetic emo algorithms». *GECCO*, pp. 1234–1245.

NISHIMURA, TAKUJI y MATSUMOTO, MAKOTO (1998). «Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator». *ACM Transactions on Modeling and Computer Simulation*, **Vol. 8(Nro. 1)**.

OSYCZKA, ANDRZEJ (2002). *Evolutionay Algorithms for Single and Multicriteria Desing Optimization*. Physica-Verlang.

PRETOLANI, D. (2000). «A directed hyper-graph model for random time dependent shortest paths». *European Journal of Operational Research*, **Vol. 123**, pp. 315–324.

REEVES, C. (1993). «Modern Heuristic Techniques for Combinatorial Problems». *Blackwell Scientific Publications*.

ROSS, SHELDON M. (2002). *Simulation*. Academic Press, 3ª edición.

RUBINSTEIN, REUVEN Y. (1981). *Simulation and the Monte Carlo Method*. Wiley Series in Probability and Mathematical Statistics.

RUDOLPH, G. (1994). «Convergence analysis of canonical genetic algoritms». *IEEE Transactions on Neural Networks*, pp. 96–101.

SCHAFFER, J. DAVID (1985). «Multiple Objective Optimization with Vector Evaluated Genetic Algorithms». En: *Proceedings of the 1st International Conference on Genetic Algorithms*, pp. 93–100. L. Erlbaum Associates Inc., Hillsdale, NJ, USA. ISBN 0-8058-0426-9.

SNYDER, T. L. y STEELE, J. M. (1995). *Probabilistic networks and network algorithms*. Elsevier.

SRINIVAS, N. y DEB, K. (1994). «Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms». *Evolutionary Computation*, **Vol. 2(Nro. 3, fall)**, pp. 221–248.

SUZUKI, J. (1993). «A construction of Bayesian networks from databases based on an MDL Principle». En: *Uncertainty in Artificial Intelligence, Proceedings of the Ninth Conference*, pp. 266–273.

Apéndice A

Archivo de Entrada

```
nodos16norm.txt
16 27
1 2 2 c 3 c
1 3 3 c 3 c
2 5 3 c 2 c
2 6 4 c 1 c
3 4 4 c 2 c
3 6 3 c 3 c
4 7 4 c 2 c
5 7 5 c 1 c
6 7 6 c 3 c
1 7 9 c 3 c
5 8 4 c 2 c
6 8 3 c 3 c
6 9 3 c 2 c
7 10 2 c 3 c
7 11 3 c 2 c
8 11 3 c 2 c
9 11 4 c 1 c
9 12 3 c 2 c
10 13 2 c 1 c
11 13 3 c 2 c
11 14 2 c 3 c
12 13 2 c 2 c
12 14 3 c 2 c
12 15 1 c 4 c
13 16 3 c 3 c
14 16 1 c 3 c
15 16 2 c 2 c
```

APÉNDICE B

CÓDIGO RUTAS CORTAS

```
#include <iostream>
#include <fstream>
#include <string >
#include <vector>
#include <algorithm>
#include "MersenneTwister.h"

#define PI 3.14159265358979323846264338328
#define MAX 1000
#define Obj 2

#define xexcel ".out"
#define xlatex ".tex"
#define xtext  ".txt"

using namespace std;

// Selecciona si es estocástico o determinístico
bool stochastic = true;

// Parámetros
int Generations        = 100,
PopCount               = 20,
RunCount               = 10;
double MutationRate    = 0.3,
CrossoverRate          = 0.5;

// Generador de números aleatorios
MTRand random;
```

```
// Archivos de entrada y salida
ofstream oexcel;
ofstream olatex;
ifstream in;
ifstream iparams;

typedef vector<int > edges;
typedef vector<edges > graph;

double Costos [ Obj ];

struct Tcosts{
        double fijo;
        char funcion;
        int parae [3];
        double parad [3];
};

struct path{
        double Cost [ Obj ];
        vector<int > Nodes;
};

typedef vector<path> population;

// Grafo de entrada
graph Graph;

// Matriz de costos
Tcosts Costs [ Obj ][MAX][MAX];
int NNodes, NEdges;
char Description [ 100 ];

// Población
population Paths;

// Conjunto de Pareto
population Front;
```

```
//-------------------------------------------------------------------
// Funciones de Distribución
//-------------------------------------------------------------------
//
// Simula una distribución exponencial
double exponencial( double lambda ){
        return -log(1 - random.randExc( ))/lambda;
}
// Simula una distribución gamma
double gamma( int n, double lambda ){
        int k=0;
        double suma;
        suma=0;
        for ( k=0;k<n ; k++)

            suma=suma+exponencial(lambda);
        return suma;
}
// Simula una distribución normal (0,1)
double normal01(){
        return ( sqrt(-2.0*log(1 - random.randExc())))

                *cos( 2.0 * PI *(1 - random.randExc()));
}
// Simula una distribución normal (media, varianza)
double normal ( double myu, double sigmacuad ){
        return ( normal01()*sqrt(sigmacuad)+myu );
}
// Simula una distribución uniforme continua ( 0,1 )
double uniformec01(){
        return random.randExc ();
}
// Simula una distribución uniforme continua (a,b)
double uniformec(double a,double b ){
        return ( uniformec01()*( b-a)+a );
}
// Simula una distribución Chi cuadrada
```

```
double chi (int grados){
        int k=grados/2;
        double x,a;
        x=gamma(k,0.5);
        if(grados %2==1){
                  a=normal01();
                  x=x+a*a;
        }
        return x;
}
// Simula una distribución poisson
int poisson( double lambda ){
    double suma;
    int x;
    suma=exponencial ( lambda );
    x=0;
    while (suma<=1){
          suma=suma+ exponencial ( lambda );
          x++;
    }
    return x;
}
// Simula una distribución hipergeométrica
int hipergeo ( int M, int K, int n ){
    int i, x;
    int fish;
    vector<int > Pbl;
    Pbl.clear ();
    for (i=0; i<K; i++)

        Pbl.push_back (1);
    for ( ;i<M;i++)

        Pbl.push_back (0);
    x=0;
    for (i=0;i<n;i++){
        fish=(int)(random.randExc()*Pbl.size());
        x=x+Pbl[fish];
```

```
        Pbl.erase(Pbl.begin()+fish);
    }
    return x;
}
// Simula una distribución geométrica
int geometrica(double p){
    double pp;
    int i,suma;
    suma=0;
    pp= random.randExc();
    while (pp>p){
          suma++;
          pp= random.randExc();
    }
    return suma;
}
// Simula una distribución binomial
int binomial(int n,double p){
    double pp;
    int i,suma;
    suma=0;
    for (i=0;i<n;i++){
        pp= random.randExc();
        if (pp<=p) suma++;
    }
    return suma;
}
// Simula una distribución uniforme discreta
int uniformed ( int n ){
    return int (n*random.randExc()+1 );
}
// Simula una distribución bernoulli
int bernuli ( double p ) {
    return binomial ( 1,p );
}
// Simula una distribución binomial negativa
int binneg ( int r, double p ) {
    double pp;
```

```
    int i, suma;
    suma=0;
    while ( r ) {
          pp = random.randExc();
          while (pp>p) {
                suma++;
                pp= random.randExc();
          }
          r--;
    }
    return suma;
}
//-------------------------------------------------------------------
// Lectura de parámetros de las funciones de distribución
//-------------------------------------------------------------------
void leerparametros ( int x, int y, int z ){
     switch ( Costs [ x ] [ y ] [ z ].funcion ) {
case 'a': case 'A':
in>> Costs [ x ] [ y ] [ z ].parad[ 0 ];
break;
case 'b': case 'B':
in>> Costs [ x ] [ y ] [ z ].parae[ 0 ];
in>> Costs [ x ] [ y ] [ z ].parad[ 0 ];
break;
case 'c': case 'C':
break;
case 'd': case 'D':
in>> Costs [ x ] [ y ] [ z ].parad[ 0 ];
in>> Costs [ x ] [ y ] [ z ].parad[ 1 ];
break;
case 'e': case 'E':
break;
case 'f': case 'F':
in>> Costs [ x ] [ y ] [ z ].parad[ 0 ];
in>> Costs [ x ] [ y ] [ z ].parad[ 1 ];
break;
case 'g': case 'G':
in>> Costs [ x ] [ y ] [ z ].parae[ 0 ];
```

```
break;
case 'h': case 'H':
in>> Costs [ x ] [ y ] [ z ].parad[ 0 ];
break;
case 'i': case 'I':
in>> Costs [ x ] [ y ] [ z ].parae[ 0 ];
in>> Costs [ x ] [ y ] [ z ].parae[ 1 ];
in>> Costs [ x ] [ y ] [ z ].parae[ 2 ];
break;
case 'j': case 'J':
in>> Costs [ x ] [ y ] [ z ].parad[ 0 ];
break;
case 'k': case 'K':
in>> Costs [ x ] [ y ] [ z ].parae[ 0 ];
in>> Costs [ x ] [ y ] [ z ].parad[ 0 ];
break;
case 'l': case 'L':
in>> Costs [ x ] [ y ] [ z ].parae[ 0 ];
break;
case 'm': case 'M':
in>> Costs [ x ] [ y ] [ z ].parad[ 0 ];
break;
case 'n': case 'N':
in>> Costs [ x ] [ y ] [ z ].parae[ 0 ];
in>> Costs [ x ] [ y ] [ z ].parad[ 0 ];
break;
case '\n': case '␣':
break;
default:
break;
}
}
//---------------------------------------------------------------------
// Lectura del archivo de entrada
//---------------------------------------------------------------------
void LoadGraph ( char*FileName ) {
int i,j,a,b;
edges Childs;
```

```
in.open( FileName );
in.getline( Description, 100 );
in >> NNodes >> NEdges;
Childs.clear( );
for( i=0; i<NNodes ; i++)

Graph.push_back ( Childs );
for ( i=0; i<NEdges ; i++) {
in >> a >> b;
a--; b--;
for ( j=0; j<Obj ; j++) {
in >> Costs [ j ] [ a ] [ b ].fijo;
in >> Costs [ j ] [ a ] [ b ].funcion;
leerparametros( j,a,b );
}
Graph [ a ].push_back (b );
}
in.close( );
cout << Description << ":" << endl;
cout << "Graph␣with␣" << NNodes << "␣nodes␣and␣"
<< NEdges << "␣edges␣loaded." << endl;
/* Latex */
olatex << Description << ":" << endl;
olatex << "Graph␣with␣" << NNodes << "␣nodes␣and␣"
<< NEdges << "␣edges␣loaded." << endl << endl;
}
//-------------------------------------------------------------------
// Impresión de la red en archivo de Excel
//-------------------------------------------------------------------
void PrintGraph ( ) {
int i,j,v;
for ( i=0; i<NNodes ; i++) {
oexcel << ( i+1) << ":";
for ( j=0; j<Graph [ i ].size ( ) ; j++) {
v = Graph [ i ] [ j ];
oexcel << "(" << (v + 1) << ",␣";
for ( int k=0; k<Obj ; k++){
oexcel << Costs [ k ] [ i ] [ v ].fijo << "␣";
```

```
}
oexcel << ")";
}
oexcel << endl;
}
oexcel << endl;
}
//-----------------------------------------------------------------
// Creación de rutas aleatorias
//-----------------------------------------------------------------
void RandomPaths ( int ini, int fin ) {
int i,v,next;
path Path;
Paths.clear ( );
for ( i=0; i<PopCount ; i++) {
v = ini;
for ( int k=0;k<Obj ; k++){
Path.Cost [ k ]=0;
}
Path.Nodes.clear ( );
Path.Nodes.push_back ( v );
while ( v!= fin ) {
next = Graph [ v ] [ random.randInt (Graph [ v ].size ( ) -1 ) ];
Path.Nodes.push_back ( next );
v = next;
}
Paths.push_back (Path );
}
olatex << "Population of size " << PopCount
<< " created." << endl;
}
//-----------------------------------------------------------------
// Costos aleatorios
//-----------------------------------------------------------------
void VarCost ( int a, int b ) {
int k;
double x;
x=0;
```

```
for ( k=0;k<Obj ; k++){
Costos [ k]=Costs [ k ] [ a ] [ b ].fijo;
if ( stochastic ) {
switch ( Costs [ k ] [ a ] [ b ].funcion ) {
case 'a': case 'A':
x=exponencial ( Costs [ k ] [ a ] [ b ].parad [ 0 ] );
break;
case 'b': case 'B':
x=gamma( Costs [ k ] [ a ] [ b ].parae [ 0 ],
Costs [ k ] [ a ] [ b ].parad [ 0 ] );
break;
case 'c': case 'C':
x=normal01 ( );
break;
case 'd': case 'D':
x=normal ( Costs [ k ] [ a ] [ b ].parad [ 0 ],
Costs [ k ] [ a ] [ b ].parad [ 1 ] );
break;
case 'e': case 'E':
x=uniformec01 ( );
break;
case 'f': case 'F':
x=uniformec ( Costs [ k ] [ a ] [ b ].parad [ 0 ],
Costs [ k ] [ a ] [ b ].parad [ 1 ] );
break;
case 'g': case 'G':
x=chi ( Costs [ k ] [ a ] [ b ].parae [ 0 ] );
break;
case 'h': case 'H':
x=poisson ( Costs [ k ] [ a ] [ b ].parad [ 0 ] );
break;
case 'i': case 'I':
x=hipergeo ( Costs [ k ] [ a ] [ b ].parae [ 0 ],
Costs [ k ] [ a ] [ b ].parae [ 1 ],
Costs [ k ] [ a ] [ b ].parae [ 2 ] );
break;
case 'j': case 'J':
x=geometrica ( Costs [ k ] [ a ] [ b ].parad [ 0 ] );
```

```
break;
case 'k': case 'K':
x=binomial ( Costs [ k ] [ a ] [ b ].parae [ 0 ],
Costs [ k ] [ a ] [ b ].parad [ 0 ] );
break;
case 'l': case 'L':
x=uniformed ( Costs [ k ] [ a ] [ b ].parae [ 0 ] );
break;
case 'm': case 'M':
x=bernuli ( Costs [ k ] [ a ] [ b ].parad [ 0 ] );
break;
case 'n': case 'N':
x=binneg ( Costs [ k ] [ a ] [ b ].parae [ 0 ],
Costs [ k ] [ a ] [ b ].parad [ 0 ] );
break;
case '\n': case ' ':
break;
default:
cout << "Funcion no encontrada ";
break;
}
Costos [ k]+=x;
}
}
}
//-----------------------------------------------------------------
// Calcula Costos de toda la población
//-----------------------------------------------------------------
void ObtainFitness ( ) {
int i,j,k,v, next;
for ( i=0; i<Obj; i++) {
Costos [ i ]=0;
}
for( i =0; i<Paths.size ( ) ; i ++) {
for ( k=0; k< Obj ; k++) {
Paths [ i ].Cost [ k ]=0;
}
v = Paths [ i ].Nodes [ 0 ];
```

```
for ( j =1; j<Paths [ i ].Nodes.size ( ) ; j++) {
next = Paths [ i ].Nodes [ j ];
VarCost ( v, next );
for ( k=0; k<Obj ; k++) {
Paths [ i ].Cost [ k]+=Costos [ k ];
}
v = next;
}
}
}
//-----------------------------------------------------------------
// Creación de conjuntos de Pareto
//-----------------------------------------------------------------
void Pareto2 ( ) {
int i, j, k, f, l;
bool band, entra;
f=Front.size()-1;
for ( i=1; i<Paths.size ( ) ; i ++) {
entra=true;
for ( k=0; k<=f ; k++) {
band=true;
for ( l =0; l<Obj ; l++){
if ( Paths[ i ].Cost [ l ]<=Front [ k ].Cost [ l ] );
else band=false;
}
if ( band ) {
Front.erase( Front.begin ()+k );
f=f -1;
k=k-1;
} else {
band=true;
for ( l =0; l<Obj ; l ++) {
if ( Paths [ i ].Cost [ l ]>=Front [ k ].Cost [ l ] );
else band=false;
}
if ( band ) entra=false;
}
}
```

```
if ( entra ) {
Front.push_back ( Paths [ i ] );
f++;
}
}
}
//-----------------------------------------------------------------
// Impresión de conjuntos de Pareto
//-----------------------------------------------------------------
void PrintPareto ( ) {
int i, j, k;
oexcel << Front.size () << endl;
for ( i =0; i<Front.size( ) ; i ++) {
oexcel << "(";
for ( j =0; j<Front [ i ].Nodes.size ( ) ; j++) {
if ( j ) oexcel << ",";
oexcel << Front [ i ].Nodes [ j ]+1;
}
oexcel << ")";
for ( k=0;k<Obj ; k++) {
oexcel << "\t" << Front [ i ].Cost [ k ];
}
oexcel << endl;
}
oexcel << endl;
// latex

olatex << "\\begin{tabular}{l";
for( k=0; k<Obj ; k++) olatex << "l";
olatex << "}" << endl;
for ( i =0; i<Front.size ( ) ; i ++) {
olatex << "(";
for ( j =0; j<Front [ i ].Nodes.size ( ) ; j++) {
if ( j ) olatex << ",";
olatex << Front [ i ].Nodes [ j ]+1;
}
olatex << ")";
for ( k=0;k<Obj ; k++) {
```

```
olatex << "\t&\t" << Front [ i ].Cost [ k ];
}
olatex << "\\\\" << endl;
}
olatex << "\\end{tabular}" << endl << endl;
}
//-------------------------------------------------------------------
// Impresión de la población
//-------------------------------------------------------------------
void PrintPaths ( ) {
int i, j, k;
// l a t e x

olatex << "\\begin{tabular}{l";
for ( k=0; k<Obj ; k++) olatex << "l";
olatex << "}" << endl;
for ( i =0; i<Paths.size ( ) ; i ++) {
olatex << "(";
for ( j =0; j<Paths [ i ].Nodes.size ( ) ; j++) {
if ( j ) olatex << ",";
olatex << Paths [ i ].Nodes [ j ]+1;
}
olatex << ")";
for ( k=0;k<Obj ; k++) {
olatex << "\t&\t" << Paths [ i ].Cost [ k ];
}
olatex << "\\\\" << endl;
}
olatex << "\\end{tabular}" << endl << endl;
}
//-------------------------------------------------------------------
// Actualización de rutas por recombinación y mutación
//-------------------------------------------------------------------
void NewRandomPaths ( int ini, int fin ) {
int i, v, next;
int ToCrossOver = ( int ) ( PopCount*(1 - MutationRate ) );
int nodes;
/* Rutas mutadas */
```

```
for ( i=ToCrossOver ; i<PopCount ; i++) {
Paths [ i ].Nodes.clear ( );
}
/* Rutas recombinadas */
for ( i =0; i<ToCrossOver ; i++) {
nodes = ( int ) ( CrossoverRate *Paths [ i ].Nodes.size ( ) );
while ( nodes ) {
Paths [ i ].Nodes.pop_back ( );
nodes--;
}
}
/* Regenerar población */
for ( i =0; i<PopCount ; i ++) {
if ( ! Paths [ i ].Nodes.size ( ) )
Paths [ i ].Nodes.push_back (ini );
v = Paths [ i ].Nodes.back ( );
while ( v!= fin ) {
next =
Graph [ v ] [ random.randInt (Graph [ v ].size ( ) -1 ) ];
Paths [ i ].Nodes.push_back ( next );
v = next;
}
}
}
//-----------------------------------------------------------------
// Lectura de parámetros
//-----------------------------------------------------------------
int LoadParams ( ) {
iparams.open ("params.cnf" );
iparams >> Generations >> PopCount >> RunCount;
iparams >> MutationRate >> CrossoverRate;
cout << "(G=" << Generations << ",P=" << PopCount
<< ",M=" << MutationRate << ",C=" << CrossoverRate
<< ")" << endl;
iparams.close ( );
}
//-----------------------------------------------------------------
// Codigo principal
```

```
//-------------------------------------------------------------------
int main ( int argc, char*args [ ] ) {
clock_t start = clock ( );
int i, run;
char basefile [ 80 ];
char archivo [ 80 ];
char outfile [ 80 ];
char opcion [ 80 ];
if ( argc <=3) {
cout << "Forma de uso: " << endl;
cout << " paths3 input output estocastico " << endl;
return 0;
}
// Leer valores de los parametros
LoadParams ( );
// Lector de parámetros de entrada
strcpy ( archivo, args [ 1 ] );
strcat ( archivo, xtext );
strcpy ( basefile, args [ 2 ] );
strcpy ( opcion, args [ 3 ] );
// Opción caso determinístico o estocástico
if ( !strcmp ( opcion, " false " ) ) stochastic = false;
else stochastic = true;
// Abrir archivos de entrada y salida
strcpy ( outfile, basefile );
strcat ( outfile, xexcel );
oexcel.open (outfile );
strcpy ( outfile, basefile );
strcat ( outfile, xlatex );
olatex.open (outfile );
LoadGraph ( archivo );
// Ciclo principal del programa
cout << "Running " << RunCount << " cases... " << endl;
oexcel << RunCount << endl;
for ( run=0; run<RunCount ; run++) {
oexcel << "Case " << ( run+1) << ":" << endl;
olatex << "Case " << ( run+1) << ":" << endl;
Front.clear ( );
```

```
RandomPaths ( 0, NNodes - 1 );
ObtainFitness ( );
PrintPaths ( );
olatex << "Running " << Generations
<< " generations..." << endl << endl;
for ( i =0; i<Generations ; i++) {
NewRandomPaths ( 0, NNodes - 1 );
ObtainFitness ( );
Pareto2 ( );
}
PrintPareto ( );
}
// Cerrar archivos de salida
oexcel.close ( );
olatex.close ( );
// Reportar tiempo de ejecución
cout << " time: " << ( clock ()- start ) / ( double )CLOCKS_PER_SEC
<< " segs " << endl;
return 0;
}
```

APÉNDICE C

MANUAL DEL PROGRAMA

En este manual se describe la forma como se puede utilizar el programa que se creó para resolver el problema de ruta corta estocástico multiobjetivo, cuyo código original se encuentra en el C.D. anexo.

§C.1. PROGRAMA PATHS3.CPP

Este programa se encarga de leer los datos desde un archivo de texto previamente existente, a partir de los cuales se obtienen los óptimos de Pareto que son impresos en dos tipos de archivos: uno que puede ser abierto en un editor de textos sencillo a partir del cual se pueden graficar los frentes de pareto para realizar un estudio y otro en latex para poder utilizar los datos de los frentes en documentos.

Este programa puede correrse desde una ventana de comandos. La forma para correr el programa es la siguiente:

```
paths3 input output estocastico
```

donde «input» se refiere al nombre del archivo de entrada. «output» es el nombre del archivo en el cual se desea obtener los resultados. La opción «estocástico» da a escoger si se desea que se resuelva de forma determinística o estocástica el problema y tiene dos opciones, que son «true» si es estocástico o «false» en caso contrario.

Si se desea resolver en forma determinística el problema se debe escribir:

```
paths3 entrada resultados false
```

donde «entrada» es el nombre del archivo de texto que contiene los datos de la red; «resultados» es el nombre de los archivos de salida (uno para abrir en .out y otro para latex).

§C.2. OBJETIVOS

El programa está hecho de tal forma que resuelve problemas de uno o más objetivos. Esta opción se puede cambiar en el código directamente, en la parte que dice como sigue:

```
#define Obj 2
```

La instrucción anterior indica que se tienen dos objetivos. Si se quisiera cambiar el número de objetivos, sólo se escribe el número correspondiente a ellos en el lugar del 2.

Como se trató de resolver problemas multiobjetivo, se eliminó la opción de un objetivo y se consideraron dos, ya que para la investigación y graficación es más comprensible, además de que es imposible graficar más de tres objetivos.

§C.3. Restricciones de las Redes

Por el tipo de problema, el usario se debe asegurar de no trabajar con redes cíclicas. Por otro lado en el programa se puso la restricción de no trabajar con redes mayores a 1000 nodos, instrucción que se representa de la siguiente manera:

```
#define MAX 1000
```

Sin embargo, si se quisiera cambiar y aumentar o disminuir dicho límite, basta con cambiar el número en la instrucción.

§C.4. Cambio en los Parámetros

Los parámetros de la forma como se ejecutará el algoritmo están especificados en un archivo llamado «params.cnf». En él se indica el número de generaciones que se desean realizar, el tama?no de la población, cuantas corridas del programa se desean, el porcentaje de mutación y el de recombinación, como se puede ver en el ejemplo siguiente:

```
1000 20 10
0.3 0.5
```

Este archivo se puede abrir en cualquier editor de textos donde se pueden hacer las modificaciones que se deseen.

§C.5. Formato del Archivo de Entrada

Dentro del problema multiobjetivo de ruta corta estocástico, se tienen muchos datos de entrada, por lo que el programa que se desarrolló lee de un archivo toda la información necesaria para poder resolver el problema. La primera línea que lee es el nombre del archivo. En la siguiente línea se tiene la descripción del grafo que se va a trabajar, leyéndose el número de nodos (m) que contiene la red y el número de arcos que unen los nodos (n). A partir de la siguiente línea, se lee en cada línea la descripción de cada uno de los arcos que forman el grafo.

Cada arco tiene un nodo origen y un nodo final, que en el archivo están representados por números, es decir el nodo del que se desea partir es el número 1 y el último nodo, al que se desea llegar, es el m-ésimo. Los datos que entran desde el archivo, son aquéllos que describen el arco, como ya se mencionó, los nodos que lo forman y los p costos, de la siguiente forma. El primer número es el nodo inicial i, el segundo es el nodo final j, luego el costo fijo del primer objetivo, una clave que tiene asignada cada distribución que debe estar seguida de los valores de sus parámetros

según la tabla 3.8.1; el costo fijo del segundo objetivo, una clave de la distribución y valores de los parámetros; el costo fijo ... y así hasta el p-ésimo objetivo.

La representación de la entrada de los arcos es como se observó en la tabla 3.8.2 donde cf_{ij} es el costo fijo de ir del nodo i al j y cv_{ij} es el costo variable de ir del nodo i al j. La siguiente es una muestra de las primeras líneas de un archivo de entrada que describe una red de 50 nodos con 226 arcos y diversos costos variables asociados a ellos.

```
nombre.txt
50 226
1 2 5 L 65 5 M 0.792617
1 3 3 M 0.611674 5 M 0.505015
1 4 1 G 10 9 A 0.709064
1 5 8 I 96 68 13 7 G 10
1 6 7 I 28 25 18 4 A 1.59396
1 7 2 C 9 G 9
```

La definición de la red siempre debe ser de la forma anterior, incluyendo ambos tipos de costos. Cuando se resuelve el problema en forma determinística, el programa no toma en cuenta los costos variables, y únicamente calcula los totales con los costos fijos.