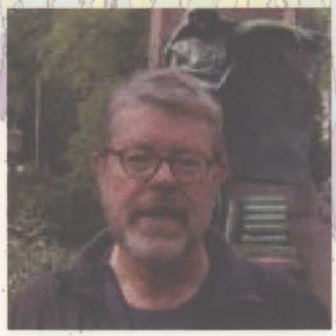
Colección CBI

BÚSQUEDA Y EXPLORACIÓN ESTOCÁSTICA

Sergio Gerardo de los Cobos Silva
John Goddard Close
Miguel Angel Gutiérrez Andrade
Alma Edith Martínez Licona

UNIVERSIDAD AUTONOMA METROPOLITANA
Casa abierta al tiempo
UNIDAD IZTAPALAPA División de Ciencias Básicas e Ingeniería

Sergio Gerardo de los Cobos Silva, es matemático por la Universidad Autónoma Metropolitana, Maestro en Ciencias por el Colegio de Postgraduados, Doctor en Ingeniería por la Universidad Nacional Autónoma de México, Doctor en Ciencias por el CIDEM y tiene un Diplomado en Estudios Avanzados por la Universitat Rovira i Virgili de España. Es profesor de la UAM-Iztapalapa. Sus áreas de interés principales son: Métodos Matemáticos de Optimización y Estadística Aplicada.



John C.H. Goddard recibió una B.Sc (1st Class Hons) de la Universidad de Londres, y un Ph.D en Matemáticas de la Universidad de Cambridge. Es Profesor en el Departamento de Ingeniería Eléctrica de la Universidad Autónoma Metropolitana. Sus intereses incluye reconocimiento de patrones y algoritmos heurísticos aplicados a problemas de optimización.

BÚSQUEDA Y EXPLORACIÓN ESTOCÁSTICA

Universidad Autónoma Metropolitana

Rector General
Dr. Enrique Pablo Alfonso Fernández Fassnacht

Secretaria General
Mtra. Iris Edith Santacruz Fabila

Unidad Iztapalapa

Rector
Dr. Javier Velázquez Moctezuma

Secretario
Dr. Óscar Comas Rodríguez

División de Ciencias Básicas e Ingeniería

Directora
Dra. Verónica Medina Bañuelos

Secretario Académico
Dr. José Antonio de los Reyes Heredia

Colección CBI

BÚSQUEDA Y EXPLORACIÓN ESTOCÁSTICA

Sergio Gerardo de los Cobos Silva

John Goddard Close

Miguel Ángel Gutiérrez Andrade

Alma Edith Martínez Licona

UNIVERSIDAD AUTONOMA METROPOLITANA
Casa abierta al tiempo
UNIDAD IZTAPALAPA División de Ciencias Básicas e Ingeniería

Primera edición, 2010

Los derechos de reproducción de esta obra pertenecen al autor
Universidad Autónoma Metropolitana

Prolongación Canal de Miramontes No. 3855,
Col. Ex Hacienda San Juan de Dios.
Delegación Tlalpan C.P. 14387 México D.F.

Universidad Autónoma Metropolitana Unidad Iztapalapa
División de Ciencias Básicas e Ingeniería

ISBN 978-607-477-239-5

Se prohibe la reproducción por cualquier medio, sin el consentimiento
de los titulares de los derechos de la obra

Impreso en México/*Printed in Mexico*

Prefacio

En este trabajo damos una presentación de algunos métodos de optimización que se utilizan en inteligencia artificial aplicada. Está dirigido para estudiantes de ingeniería, física y matemáticas aplicadas, computación, así como para cualquier área donde se necesite resolver problemas difíciles de optimización. Las materias a las que se enfoca el presente trabajo son: optimización, investigación de operaciones, programación matemática, análisis de algoritmos e inteligencia artificial, entre otras. Para la lectura de este texto no se requiere de conocimientos especiales, hemos tratado de que la obra sea autocontenida.

El material puede cubrirse durante al menos un curso trimestral intensivo, o en un curso semestral. Como consideramos que la motivación es muy importante en el proceso de aprendizaje, hemos planteado problemas que se resuelven completamente indicando los diferentes pasos que llevan a su solución. En cada capítulo van precedidos de una presentación de sus temas. Aspiramos con ello a fijar la atención de nuestros lectores sobre algunos temas fundamentales que se abordan para solucionar problemas de optimización y sobre los puntos más relevantes de las técnicas tratadas.

La finalidad del presente texto es, en última instancia, contribuir a que los lectores tomen conciencia de la problemática de las cuestiones abordadas y de la necesidad de contar con herramientas que les permitan atacar problemas difíciles o muy difíciles que se presentan con gran frecuencia en el ámbito de la inteligencia artificial y sus aplicaciones. Nuestro trabajo pretende también, en el campo que le corresponde, ayudar a los alumnos a que aprendan a distinguir entre soluciones óptimas y soluciones de calidad que no necesariamente son óptimas pero que, en general, son muy buenas si se aplican las técnicas adecuadamente.

Los capítulos tienen referencias de diverso género: teórico, experimental o aplicaciones prácticas. Es aquí donde el alumno puede explorar para aumen-

tar sus conocimientos a fin de que la lectura deje el menor número de lagunas posible y sea, por tanto, más fructífera. Deseamos, por último, expresar que entregamos un trabajo del cual esperamos que los lectores compartan con nosotros la opinión de que es útil y fácil de leer.

El trabajo se agrupa en siete capítulos que tratan sobre métodos de búsqueda y exploración heurística para problemas de optimización. En el capítulo uno, damos una introducción al material tratado por el libro. En el capítulo dos, se proporciona la notación que utilizaremos así como a algunos conceptos básicos importantes. En el capítulo tres, nos referimos a una colección de problemas de tipo NP y algunas funciones continuas. En el capítulo cuatro, se propone el método de recocido simulado y se presentan algunos ejemplos utilizando problemas combinatorios como son el del conjunto independiente y el de asignación cuadrática. En el capítulo cinco, se presenta la búsqueda tabú, y se ejemplifica, paso a paso, mediante un problema de calendarización, posteriormente se presenta un problema de aplicación referente a la recolección de desechos sólidos de plataformas petroleras y por último, se indica la implementación de listas tabú de tipo dinámico utilizando el problema de asignación cuadrática. En el capítulo seis, introducimos el tema de optimización por enjambre de partículas y se proporcionan dos ejemplos de aplicación: el problema de inventario multiproducto y el problema de regresión no lineal; además introducimos al tema de optimización por el método hormiga y se implementa para el problema del agente viajero. En el capítulo siete, hacemos una presentación de los algoritmos genéticos, revisando mediante ejemplos, incluyendo al problema del agente viajero, diferentes características de estos, posteriormente se introduce a las estrategias evolutivas, la que se ejemplifica con dos ejemplos: uno para la maximización de una función y otro ejemplo, de clasificación mediante una red neuronal utilizando un programa evolutivo.

Agradecimientos

Deseamos expresar nuestro reconocimiento y gratitud en la realización de este libro a la Secretaría de Educación Pública por el apoyo recibido a través de su Programa de Mejoramiento del Profesorado (PROMEP). En particular agradecemos el apoyo prestado al Cuerpo Académico Consolidado de Inteligencia Artificial y sus Aplicaciones, al cual pertenecen los autores.

A la Dirección de la División de Ciencias Básicas e Ingeniería de la UAM-Iztapalapa por las facilidades brindadas y la ayuda económica que nos proporcionaron en especial a la Dra. Verónica Medina Bañuelos, Directora de la División y al Secretario Académico, Dr. José Antonio de los Reyes Heredia.

Al Departamento de Ingeniería Eléctrica de la UAM-Iztapalapa por todo el apoyo y facilidades brindadas, en especial al Jefe del Departamento Dr. Víctor Manuel Ramos Ramos.

También, deseamos expresar nuestro agradecimiento a los árbitros anónimos por el tiempo que se tomaron en leer todo el documento y sus valiosos comentarios y sugerencias que hicieron que el libro se enriqueciera.

Finalmente, agradecemos a The M.C. Escher Company-Holland por permitirnos usar la imagen "Reptiles" en la portada del libro.[1]

[1]M.C. Escher's "Reptiles" ©2009 The M.C. Escher Company-Holland. All rights reserved. www.mcescher.com.

Índice General

Prefacio **3**

Agradecimientos **5**

Índice de Figuras **11**

Índice de Tablas **15**

1 Introducción **19**

2 Conceptos y Definiciones Preliminares **23**
2.1 El Problema de Optimización 23
2.2 Optimización Continua 24
2.3 Optimización Combinatoria 25
2.4 Heurísticas y Metaheurísticas 30

3 Problemas de Optimización **33**
3.1 Ejemplos de Problemas NP 33
3.1.1 Problemas P y NP 33
3.1.2 Resolviendo Problemas NP. 35
3.1.3 Problema de Asignación Cuadrática 36
3.1.4 Problema de Conjunto Independiente 39
3.1.5 Problema de Coloración Mínima 40
3.1.6 Problema de Empaquetamiento 41
3.1.7 Problema de Cubrimiento de Vértices 41
3.1.8 Problema del Agente Viajero 42
3.1.9 Problema de Clan Máximo. 42
3.1.10 Problema de Corte Máximo. 42
3.1.11 Problema de Satisfacibilidad 43

3.1.12 Problema de la Mochila 44
3.1.13 Cuadrados Mágicos 44
3.1.14 El Problema del Sudoku 45
3.1.15 Problema de Asignación Tridimensional Axial 46
3.1.16 Problema de Aprendizaje en Perceptrones Multicapa . 46
3.2 Problemas Continuos 49
3.2.1 La función de Ackley 49
3.2.2 La función RCOS de Branin 50
3.2.3 La función de Easom 51
3.2.4 La función de la esfera 51
3.2.5 La función de Goldstein-Price 52
3.2.6 La función de Griewangk 53
3.2.7 La función de Langermann 54
3.2.8 La función de Michalewicz 56
3.2.9 La función de Rastrigin 57
3.2.10 La función de Rosenbrock 58
3.2.11 La función de Schaffer F6 59
3.2.12 La función de Schwefel 60
3.2.13 Las seis jorobas del camello 61

4 El Algoritmo de Recocido Simulado 63
4.1 Recocido Simulado 63
4.1.1 El Proceso de Recocido de un Sólido 64
4.1.2 El Algoritmo de Recocido Simulado 66
4.1.3 Aspectos Generales del Algoritmo 69
4.2 Ejemplo: El Problema de Conjunto Independiente 71
4.2.1 Algoritmo de Recocido Simulado 71
4.2.2 Descripción del Algoritmo 71
4.2.3 Experiencia Computacional 72
4.2.4 Análisis Comparativo con un Método Exacto 74
4.3 Ejemplo: Problema de Asignación Cuadrático 77
4.3.1 Algoritmo de Recocido Simulado 77
4.3.2 Algoritmo de Búsqueda Local 81
4.3.3 Análisis Comparativo de los Métodos 81
4.4 Ejemplo: Sudoku 86
4.4.1 El Problema del Sudoku 86
4.4.2 Implementación de la Técnica de Recocido Simulado al Sudoku 88

4.4.3 Algoritmo de Recocido Simulado para el Sudoku 96
4.4.4 Resultados 96
4.4.5 Conclusiones 98
4.5 Ejercicios 99

5 Búsqueda Tabú 105
5.1 Introducción 105
5.2 Algoritmo de Búsqueda Tabú 108
5.3 Ejemplo: Problema de Calendarización 120
5.3.1 Algunas Observaciones para la Implementación 122
5.3.2 Memoria de Plazo Intermedio y Largo 131
5.3.3 Intensificación y Diversificación Regional. 132
5.3.4 Criterios de Aspiración 137
5.4 Ejemplo: Recolección de Desechos Sólidos 138
5.4.1 Objetivo 139
5.4.2 Información 139
5.4.3 Solución y Resultados del Ejemplo 140
5.4.4 Conclusiones del Ejemplo 142
5.5 Ejemplo: Problema de Asignación Cuadrática 143
5.5.1 Elementos de la BT para el QAP 144
5.5.2 Estrategia Básica 144
5.5.3 Manejo Dinámico de la Lista Tabú 147
5.5.4 Experiencia Computacional 148
5.6 Conclusiones 149
5.7 Lineamientos de BT para Problemas Continuos 151
5.7.1 Detalles de Implementación 151
5.8 Ejercicios 152

6 Inteligencia de Enjambre 159
6.1 Optimización por Enjambre de Partículas 160
6.1.1 Introducción 161
6.1.2 Ejemplo: Optimización de la función de Rosenbrock . . 165
6.1.3 Ejemplo: Problemas de Inventarios 167
6.1.4 Ejemplo: Regresión No Lineal. 171
6.1.5 Ejercicios 174
6.2 Método Hormiga 176
6.2.1 Introducción 176
6.2.2 Generalidades del Sistema Hormiga 177

6.2.3 Ejemplo: Agente Viajero. 179
6.2.4 Ejercicios 188

7 Algoritmos Genéticos y Estrategias Evolutivas 189
7.1 Introducción 189
7.2 Descripción General de los AG y la ES 190
7.3 Algoritmos Genéticos 191
7.3.1 Operadores Genéticos 192
7.3.2 Algoritmo Genético Básico 198
7.3.3 Elitismo 199
7.3.4 Ejemplo: Optimización de funciones reales 201
7.3.5 Ejemplo: Optimización de funciones reales de varias variables 205
7.3.6 Ejemplo: Agente Viajero. 209
7.4 Estrategias Evolutivas 213
7.4.1 Algoritmos (μ + λ)-ES y (μ,λ)-ES 214
7.4.2 Algoritmos (1 + 1)-ES 215
7.4.3 Ejemplo: Optimización de funciones reales con un algoritmo (1+1)-ES 216
7.4.4 Adaptación determinista de los parámetros 218
7.4.5 Ejemplo: Optimización de funciones reales con un algoritmo (1+1)-ES auto-adaptivo. 219
7.4.6 Adaptación aleatoria de los parámetros de la mutación 220
7.4.7 Múltiples padres 221
7.5 Programa Evolutivo de Cromosomas de Longitud Variable . . 222
7.5.1 Aprendizaje en los perceptrones multicapa (PMC) . . . 222
7.5.2 Descripción del programa evolutivo de cromosomas de longitud variable 224
7.5.3 Ejemplo: Clasificación mediante un PMC definido por un programa evolutivo 229
7.6 Observaciones Finales 239
7.7 Ejercicios 241

Bibliografía 245

Índice de Materias 257

Indice de Figuras

2.1 Función de Shubert. 2 5
2.2 Algoritmo de Búsqueda Local en pseudo-código 29

3.1 Gráfica con 8 vértices y 14 aristas. 40
3.2 Sudoku. 45
3.3 Solución válida al Sudoku de la Figura 3.2. 46
3.4 Un perceptrón multicapa. 47
3.5 Procesamiento realizado en un nodo en una capa oculta o de salida. 48
3.6 La función de Ackley en dos dimensiones. . 50
3.7 La función rcos de Branin. 51
3.8 La función Easom. 52
3.9 La función de De Jong en dos dimensiones. 53
3.10 La función Goldstein-Price. 54
3.11 La función de Griewangk en dos dimensiones. 55
3.12 La función de Langermann en dos dimensiones con m = 5. 56
3.13 La función de Michalewicz en dos dimensiones para $m = 1, 5, 10$. 57
3.14 La función de Rastrigin en dos dimensiones. 58
3.15 El valle de Rosenbrock en dos dimensiones. 59
3.16 La función de Schaffer. 60
3.17 La función de Schwefel. 61
3.18 La función de las seis jorobas del camello. 62

4.1 Algoritmo de Recocido Simulado en pseudocódigo 68
4.2 Algoritmo de Recocido Simulado para el Problema de Conjunto Independiente Máximo. 73

4.3 Algoritmo de Recocido Simulado para el Problema de Asignación Cuadrática. 79
4.4 Algoritmo de Búsqueda Local para el Problema de Asignación Cuadrática. 82
4.5 Sudoku. 87
4.6 Resolviendo el Sudoku. 88
4.7 Resolviendo el Sudoku (continuación). 89
4.8 Solución válida al Sudoku de la Figura 4.5. 89
4.9 Llenado inicial de la primera subcuadrícula. 91
4.10 Solución inicial obtenida por el procedimiento. 91
4.11 Solución vecina de la inicial. 93
4.12 Algoritmo de Recocido Simulado para el Sudoku. 97

5.1 Gráfica del ejemplo de calendarización. 107
5.2 Estructura básica de vecindad. 110
5.3 Representación gráfica de un punto tabú. 111
5.4 Proceso de intensificación regional. 112
5.5 Esquema 1 Selección del mejor candidato admisible. 115
5.6 Esquema 2 Componente de memoria de término corto de la búsqueda tabú. 116
5.7 Diversificación regional. 117
5.8 Representación gráfica de todos los posibles valores del ejemplo. 121
5.9 Región Marina de la zonda de Campeche. 139
5.10 Región Marina de la zonda de Campeche puntos considerados en el ejemplo. 141
5.11 Trayectorias óptimas (de costo mínimo). 142

6.1 Enjambre (subconjunto de puntos en el conjunto factible). . . 161
6.2 Posición de la i-ésima partícula y componentes de movimiento. 162
6.3 Algoritmo de PSO. 163
6.4 Curvas de nivel y vectores de velocidad del ejemplo. 165
6.5 Población inicial y después de 100, 200 y 300 iteraciones del ejemplo. 166
6.6 Ilustración de la presencia de mínimos locales. 173
6.7 Algoritmo General de AS. 180

7.1 Método de selección con la ruleta. 195
7.2 Ejemplo de un solo punto de cruzamiento. 196

7.3 Cruzamiento con dos puntos. 197
7.4 Un algoritmo genético. 199
7.5 Pseudocódigo de un algoritmo genético. 200
7.6 $f(x) = x + 5 * sin(3 * x) + 8 * cos(5 * x)$. 201
7.7 Población inicial. 205
7.8 Valores obtenidos en la primera generación. 206
7.9 Evolución de las mejores soluciones encontradas por el algoritmo. 206
7.10 Valor máximo. 207
7.11 $f(x, y) = [x + 5 * sin(3 * x) + 8 * cos(5 * x) + y + 5 * sin(3 * y) + 8 * cos(5 * y)]/2$. 208
7.12 Los mejores valores obtenidos en una corrida de 200 iteraciones. 209
7.13 Los mejores valores obtenidos en 10 corridas del algoritmo genético. 209
7.14 Los porcentajes asignados a los mismos seis cromosomas de la tabla 7.1, con s=0.92. 212
7.15 Ciudades escogidas para el ejemplo con el recorrido más corto. 213
7.16 Los mejores valores y los valores promedios por cada iteración. 214
7.17 La desviación estandar en cada iteración. 214
7.18 Un algoritmo $(\mu + \lambda)$-ES. 215
7.19 Pseudocódigo para un algoritmo $(\mu + \lambda)$-ES. 216
7.20 Gráfica de la función en el intervalo [0,3]. 218
7.21 Evolución de la desviación estándar iniciando con $\sigma = 0.01$. . . 220
7.22 Representación de un PMC mediante un cromosoma en el programa evolutivo. 225
7.23 Estructuras de datos del programa evolutivo. 226
7.24 Algoritmo Principal del Programa Evolutivo. 230
7.25 Gráfica de las clases del ejemplo de IRIS. 231
7.26 Ejemplo de la base de datos Peterson Barney. 234

Índice de Tablas

3.1 Valores aproximados de algunas funciones. 35

4.1 Tiempos promedio de ejecución de recocido simulado para el problema de conjunto independiente. 74
4.2 Tiempos promedio de ejecución del algoritmo exacto para el problema de conjunto independiente. 75
4.3 Comparación de los algoritmos que resuelven el problema de conjunto independiente con respecto a su eficiencia. 76
4.4 Tiempos de ejecución del algoritmo de recocido simulado para el problema de asignación cuadrática. 80
4.5 Resultados obtenidos para los problemas de Nugent por el algoritmo de recocido simulado. 80
4.6 Tiempos de ejecución del algoritmo de búsqueda local para el problema de asignación cuadrática. 83
4.7 Resultados obtenidos para los problemas de Nugent por el algoritmo de búsqueda local. 83
4.8 Comparación de los algoritmos para resolver el QAP con respecto al tiempo de ejecución y a su eficiencia. 85
4.9 Comparación entre los algoritmos heurístico, recocido simulado y búsqueda local con respecto a la dependencia de la solución inicial. 85
4.10 Penalizaciones asociadas a la solución inicial. 92
4.11 Estadísticas obtenidas en la solución de los Sudokus. 98

5.1 Restricciones tabú y atributos para movimientos de intercambio. 114
5.2 Condiciones del ejemplo. 124
5.3 Costos del ejemplo. 124
5.4 Punto inicial y primera iteración. 126

5.5 Lista tabú e historial de la primera iteración. 126
5.6 Segunda iteración. 127
5.7 Lista tabú e historial de la segunda iteración. 127
5.8 Tercera iteración. 127
5.9 Lista tabú e historial de la tercer iteración. 127
5.10 Cuarta iteración. 128
5.11 Lista tabú e historial de la cuarta iteración. 128
5.12 Quinta iteración. 128
5.13 Lista tabú e historial de la quinta iteración. 128
5.14 Sexta iteración. 129
5.15 Lista tabú e historial de la sexta iteración. 129
5.16 Décima iteración. 129
5.17 Lista tabú e historial de la décima iteración. 130
5.18 Iteraciones de la ejecución 1. 133
5.19 Iteraciones de la ejecución 2. 134
5.20 Matriz de frecuencias de la ejecución 2. 134
5.21 Iteraciones del ejemplo 1. 135
5.22 Iteraciones del ejemplo 2. 136
5.23 Iteraciones del ejemplo 3. 137
5.24 Distancias de las plataformas a Cd. del Carmen (Kms.). . . . 140
5.25 Distancias de las plataformas a Dos Bocas (Kms.). 141
5.26 B.T: Búsqueda Tabú, B.B: Branch and Bound, G.A.T: Trayec - toria del ángulo mayor. 143
5.27 Resultados comparativos para el problema de asignación cuadrático. 15

6.1 Datos del ejemplo numérico tomado de [35]. 170
6.2 Distancias entre ciudades para el ejemplo. 179
6.3 Matrices de feromonas en la construcción de la ruta (5-6-1- 4-2-3) . 181
6.4 Matriz de rutas y sus costos para la primera iteración de la corrida 1. 182
6.5 Matriz de rutas y sus costos para la segunda iteración de la corrida 1. 182
6.6 Matriz de feromonas en la construcción de la ruta (3-6-5-1-4-2) . 183
6.7 Matriz de rutas y sus costos para la tercera iteración de la corrida 1. 183
6.8 Matriz de rutas y sus costos para la primera iteración de la corrida 2. 184

6.9 Matriz de rutas y sus costos para la segunda iteración de la corrida 2. 184
6.10 Matriz de rutas y sus costos para la tercera iteración de la corrida 2. 184
6.11 Matriz de rutas y sus costos para la primera iteración de la corrida 3. 185
6.12 Matriz de rutas y sus costos para la segunda iteración de la corrida 3. 185
6.13 Matriz de rutas y sus costos para la tercera iteración de la corrida 3. 185
6.14 Matriz de rutas y sus costos para la cuarta iteración de la corrida 3. 186
6.15 Matriz de rutas y sus costos para la quinta iteración de la corrida 3. 186
6.16 Matriz de rutas y sus costos para la sexta iteración de la corrida 3. 186
6.17 Matriz de rutas y sus costos para la séptima iteración de la corrida 3. 187
6.18 Matriz de rutas y sus costos para la octava iteración de la corrida 3. 187

7.1 Datos del resultado del ejemplo. 196
7.2 Resultados después de la primera generación. 196
7.3 Los mejores aptitudes encontrados con los diferentes valores de σ. 218
7.4 Los mejores aptitudes encontrados con los diferentes valores de σ. 220
7.5 Características relevantes de las bases de datos. 237
7.6 Número de ejemplos de las bases de datos. 238
7.7 Tiempos de ejecución del programa evolutivo. 238
7.8 Porcentajes de acierto del programa evolutivo. 239

Capítulo 1

Introducción

Los problemas de optimización se dividen de manera natural en dos categorías: problemas de optimización con variables *continuas* y problemas de optimización con variables *discretas*. A estos últimos se les llama *problemas de optimización combinatoria*. En problemas de tipo continuo, se busca la solución sobre un conjunto de números reales con ciertas propiedades de continuidad y generalmente de convexidad; en los problemas de optimización combinatoria, se busca la solución sobre un conjunto finito o infinito numerable de objetos para los que se pueda definir una función que evalúe la "calidad" de cada objeto. Estas dos clases de problemas tienen diferentes retos y los métodos para resolverlos son distintos.

Uno de los problemas computacionales a resolver en optimización combinatoria es la *explosión combinatoria*. La explosión combinatoria se encuentra en situaciones donde las elecciones están compuestas secuencialmente, es decir, dado un conjunto de elementos se pueden obtener diferentes arreglos ordenados de éstos, permitiendo una vasta cantidad de posibilidades. Situaciones de este tipo ocurren en problemas de inversión financiera, manejo de inventarios, diseño de circuitos integrados, manejo de recursos hidráulicos, mantenimiento de sistemas, etc.

Una característica recurrente en los problemas de optimización combinatoria es el hecho que son muy "fáciles" de entender y enunciar, pero generalmente son "difíciles" de resolver en tiempo razonable. Podría pensarse que la solución de un problema de optimización combinatoria se restringe únicamente a buscar de manera exhaustiva el valor máximo o mínimo en un conjunto finito de posibilidades y que, usando una computadora veloz,

el problema carecería de interés matemático, sin pensar por un momento, en el tamaño de este conjunto. En la mayoría de los casos, este conjunto de posibilidades crece de manera alarmante conforme crece el tamaño de la instancia del problema. Es frecuente que los problemas de este tipo tengan $n!$ (n factorial) o más soluciones factibles (donde n es el tamaño de la instancia). Ahora, si una computadora pudiera ser programada para examinar soluciones a razón de un billón de soluciones por segundo, la computadora terminaría su tarea, para $n = 23$ en alrededor de 820 años, para $n = 24$ en alrededor de 19,674 años y así sucesivamente. Por lo que no tiene sentido resolver de esa forma un problema si al interesado no le alcanza la vida para ver la respuesta.

Bajo las consideraciones anteriores, el interés de la optimización combinatoria pasa a ser el de desarrollar algoritmos para los cuales el número de etapas computacionales elementales sea aceptablemente pequeño. Equivalentemente, que encuentren la solución en un tiempo razonable o como suele decirse (en términos formales) en "tiempo polinomial"; esto es, los pasos computacionales elementales es una función polinomial del número de datos.

Durante muchos años se han hecho esfuerzos para resolver problemas de optimización combinatoria, hay problemas para los que existen algoritmos rápidos y eficientes; por ejemplo, el problema de programación lineal, flujo en redes, acoplamiento, etc. Sin embargo, existen muchos problemas de optimización combinatoria para los que no se conocen algoritmos eficientes que los resuelvan cuando el tamaño del problema es mediano o de gran escala; por ejemplo, el problema del agente viajero, el problema de asignación cuadrático, el problema de particionamiento, etc. Estos últimos problemas tienen más de 50 años de ser atacados, en la mayoría de los casos, por algoritmos desarrollados especialmente para el problema específico y usando una diversidad de técnicas tales como planos de corte, ramificación y acotamiento, enumeración implícita, programación dinámica, relajación lagrangeana, entre otras, así como combinaciones de las técnicas antes mencionadas. Desafortunadamente, no se ha podido dar soluciones eficientes. También existe la necesidad de producir métodos de aplicabilidad general y flexible para problemas de optimización combinatoria que obtengan soluciones en un tiempo razonable.

En el estudio de la existencia de algoritmos que permitan encontrar la solución buscada en un tiempo polinomial y la construcción de ellos, cuando

es posible, es muy importante el conocimiento de las propiedades y estructura matemática del problema. En particular, la teoría de Gráficas permite, en muchos casos, el estudio de esta estructura y al aprovechar sus propiedades es posible construir los algoritmos buscados, pero para otros problemas, no siempre resulta fácil aprovechar la estructura. Hay problemas lineales, como los de flujo en redes y acoplamiento, que pueden resolverse de manera muy eficiente; sin embargo, estos problemas están aparentemente muy relacionados con otros problemas que se consideran "intratables". Como ejemplo, los problemas de trayectoria más corta en una gráfica y de acoplamiento, en donde se conocen algoritmos $O(n^2)$ que los resuelven; en contraste con el problema del agente viajero, el cual resuelve la trayectoria más corta para visitar el conjunto de todos los vértices de una red exactamente una vez, que como es bien conocido, es un problema en la denominada clase NP-Dura cuyos problemas son ampliamente considerados irresolubles por algoritmos polinomiales. Este fino límite entre problemas "fáciles" y "difíciles" es un fenómeno recurrente en los problemas de optimización combinatoria.

Para los problemas de optimización en la clase NP-Dura, a la fecha, no se conocen algoritmos que los resuelvan en tiempo polinomial. Es por esto que, en las últimas décadas, se han desarrollado algoritmos de tipo heurístico o estocástico para resolver instancias grandes de problemas que pertenecen a esta clase. Estos algoritmos, no necesariamente encuentran la mejor solución al problema, pero en general se pueden obtener "buenas soluciones" cuando se aplican, aquí se entiende "buena solución" en términos de cercanía al valor óptimo. La ventaja principal de este tipo de algoritmos es que son rápidos y corren en un tiempo polinomial.

Capítulo 2

Conceptos y Definiciones Preliminares

Podemos diferenciar básicamente dos tipos de optimización: la optimización discreta y la optimización continua. La optimización discreta se tiene cuando el dominio S de la función objetivo es un conjunto discreto, en cambio, se habla de optimización continua cuando el dominio S de la función objetivo es un conjunto continuo.

La búsqueda local constituye una clase interesante de los algoritmos estocásticos y está basada en el mejoramiento paso a paso de la función de costo al explorar las vecindades de soluciones cercanas. El uso del algoritmo de búsqueda local presupone la definición de una solución, una función de costo y una estructura de vecindades. A continuación, se dan algunas definiciones y ejemplos de las mismas.

2.1 El Problema de Optimización

Definición 2.1. *A la función $f : S \longrightarrow \Re$, donde $S \subseteq \Re^n$, la denominaremos función costo/beneficio o función objetivo y al conjunto S lo llamaremos conjunto factible o conjunto de soluciones posibles.*

El problema general de optimización consiste en:

> Optimizar (minimizar o maximizar) una función objetivo, y en el caso de minimización, por ejemplo, utilizamos la notación:

$$\underset{x \in S}{Minimizar} \quad f(x)$$

2.2 Optimización Continua

Para el caso de minimización en optimización continua, el problema es encontrar $x_{opt} \in S$ que satisfaga:

$$f(x_{opt}) \leq f(x) \text{ para toda } x \in S, \tag{2.1}$$

en el caso de maximización, la $x_{opt} \in S$ que satisfaga:

$$f(x_{opt}) \geq f(x) \text{ para toda } x \in S. \tag{2.2}$$

A la solución x_{opt} se le llama una *solución globalmente óptima* y $f_{opt} = f(x_{opt})$ denota el costo óptimo, mientras que S_{opt} denota el conjunto de soluciones óptimas.

Ejemplo: Función de Shubert

La función de Shubert está definida por:

$$f(x) = cos(2x+1) + 2cos(3x+2) + 3cos(4x+3) + 4cos(5x+4) + 5cos(6x+5)$$

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

Se muestra la función en la Figura 2.1 donde se aprecian tres mínimos y tres máximos globales en el intervalo.

Definición 2.2. *Dado un problema de optimización continua definimos una vecindad para $x \in S$ con radio $r \in \Re^+$ como:*

$$B_r(x) = \{y \in S | \ ||x - y|| < r\}$$

donde $|| \cdot ||$ es alguna norma definida en S.

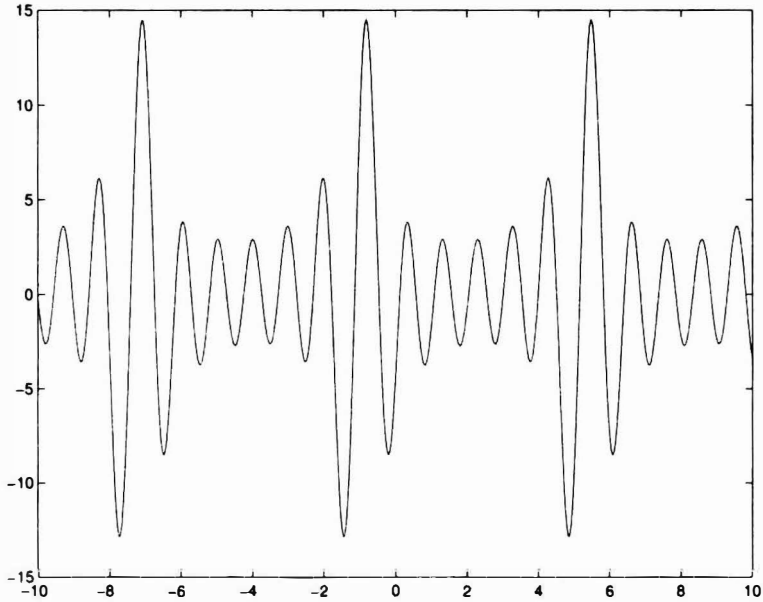


Figura 2.1: Función de Shubert.

Definición 2.3. *Sea un problema de optimización continua y $S_{\overline{x}}$ una vecindad de $\overline{x}$, entonces $\overline{x}$ se llama una solución óptima local o simplemente un óptimo local con respecto a $S_{\overline{x}}$ si $\overline{x}$ es mejor que o igual a, todas sus soluciones vecinas con respecto a la función costo. Específicamente, en el caso de minimización, $\overline{x}$ se llama solución mínima local o simplemente un mínimo local sí*

$$f(\overline{x}) \leq f(y), \text{ para toda } y \in S_{\overline{x}} \tag{2.3}$$

y en el caso de maximización, $\overline{x}$ se llama una solución máxima local o simplemente un máximo local sí

$$f(\overline{x}) \geq f(y), \text{ para toda } y \in S_{\overline{x}} \tag{2.4}$$

2.3 Optimización Combinatoria

Definición 2.4. *Una instancia de un problema de optimización combinatoria puede formalizarse como una pareja (S, f), donde S denota el conjunto finito*

de todas las soluciones posibles y f es una función real, la función de costo o función objetivo, mapeo definido por

$$f : S \rightarrow \mathbb{R} \tag{2.5}$$

Para el caso de minimización, el problema es encontrar $i_{opt} \in S$ que satisfaga:

$$f(i_{opt}) \leq f(i) \text{ para toda } i \in S \tag{2.6}$$

en el caso de maximización, la $i_{opt} \in S$ que satisfaga

$$f(i_{opt}) \geq f(i) \text{ para toda } i \in S \tag{2.7}$$

A la solución i_{opt} se le llama una *solución globalmente óptima* y $f_{opt} = f(i_{opt})$ denota el costo óptimo, mientras que S_{opt} denota el conjunto de soluciones óptimas. Un problema de *optimización combinatoria* es un conjunto I de instancias.

> **Ejemplo: El Problema del Agente Viajero.** Considere n ciudades y una matriz d_{pq} de orden $n \times n$, cuyos elementos denotan la distancia entre cada par (p,q) de ciudades. Se define un *recorrido* como una trayectoria cerrada que visita cada ciudad exactamente una vez. El problema es encontrar el recorrido con longitud mínima.

En este problema, una solución está dada por una permutación cíclica $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$, donde $\pi(k)$ denota la ciudad a visitar después de la ciudad k, con $\pi^l(k) \neq k, l = 1, 2, \ldots, n-1$ *y* $\pi^n(k) = k$, para toda k. Aquí $\pi^l(k)$ se entiende por la aplicación de l veces la permutación π. Cada solución corresponde a un recorrido. El espacio de soluciones está dado por:

$S = \{$*todas las permutaciones* π cíclicas de las n ciudades$\}$

y la función de costo se define por:

$$f(\pi) = \sum_{i=1}^{n} d_{i,\pi(i)}$$

es decir, $f(\pi)$ da la longitud del recorrido correspondiente a π. Además, se tiene que la cardinalidad de S es $|S| = (n-1)!$.

En la Definición 2.4 se ha distinguido entre un *problema* y una *instancia* del problema. De manera informal, una instancia está dada por los "datos de entrada" y la información suficiente para obtener una solución mientras que un problema es una colección de instancias del mismo tipo.

En general, un *algoritmo exacto* es un procedimiento paso a paso que resuelve un problema. Se dice que un algoritmo *resuelve* un problema, si puede aplicarse a cualquier instancia y siempre garantiza una solución. Para este caso un algoritmo exacto es aquel que al aplicarse a una instancia (S, f) obtiene $i_{opt} \in S_{opt}$.

Definición 2.5. *Sea (S, f) una instancia de un problema de optimización combinatoria. Entonces una estructura de vecindades es un mapeo:*

$$N : S \rightarrow 2^S \tag{2.8}$$

que define para cada solución $i \in S$ un conjunto $S_i \subset S$ de soluciones que son "cercanas" a i en algún sentido. El conjunto S_i se llama vecindad de la solución i y cada $j \in S_i$ se llama un vecino de i. Además se supone que $j \in S_i \Leftrightarrow i \in S_j$.

En el problema del agente viajero una estructura de vecindades N_k, llamada k-cambios, define para cada solución i una vecindad S_i que consiste del conjunto de soluciones que pueden obtenerse a partir de la solución i cambiando k aristas del recorrido correspondiente a la solución i y remplazándolos por otras k aristas de tal manera que nuevamente se obtenga un recorrido.

Si $k = 2$, entonces N_2 es la estructura de vecindades de 2-cambios. El 2-cambios $N_2(p, q)$ puede verse como invertir el orden en el que las ciudades p y q se visitan en el recorrido. Si el mecanismo 2-cambios $N_2(p, q)$ cambia una solución i en una solución j, entonces las permutaciones cíclicas π_i y π_j están relacionadas de la siguiente manera. Sea k denote el entero tal que $\pi_i^k(p) = q$, $1 < k < n$, entonces se tiene:

$$\pi_j(p) = \pi_i^{-1}(q) = \pi_i^{k-1}(p),$$

$$\pi_j(\pi_i(p)) = q = \pi_i^k(p),$$

$$\pi_j(\pi_i^r(p)) = \pi_i^{r-1}(p), \quad r = 2, \ldots, k-1,$$

$$\pi_j(s) = \pi_i(s), \quad \text{en} \quad \text{otro} \quad \text{caso}. \tag{2.9}$$

así,

$$S_i = \{j \in S \mid \pi_j \text{ se obtiene de } \pi_i \text{ por 2-cambios}\} \tag{2.10}$$

y

$$|S_i| \equiv \Theta = (n-1)(n-2) \quad \text{para} \quad \text{toda } i. \tag{2.11}$$

Además, cada solución j puede obtenerse a partir de cualquier solución i por n-2, 2-cambios, es decir, para toda $i, j \in S$ existe una sucesión $l_0, l_1, \ldots, l_{n-2} \in S$ tal que:

$$\pi_{l_0} = \pi_i \ y \ \pi_{l_{n-2}} = \pi_j \tag{2.12}$$

y $\pi_{l_{k+1}}$ se obtiene desde $\pi_{l_k}, k = 0, \ldots, n-3$, por un $N_2(p, q)$ cambios, donde

$$p = \pi_j^k(1) \ y \ q = \pi_{l_k}(\pi_j^{k+1}(1)) \tag{2.13}$$

Definición 2.6. *Sea (S, f) una instancia de un problema de optimización combinatoria, N una estructura de vecindades. Entonces un mecanismo de generación es un medio para seleccionar una solución j de la vecindad S_i de la solución i.*

Dada una instancia de un problema de optimización combinatoria y una estructura de vecindades, el algoritmo de búsqueda local es un algoritmo que itera sobre un número de soluciones. Comienza con una solución inicial, que a menudo se genera aleatoriamente, después se aplica un mecanismo de generación que continuamente trata de encontrar una mejor solución en la vecindad de la solución actual, esto es, una solución con menor costo. Si se encuentra una solución con estas características, la solución actual se reemplaza por esta solución. De otra manera, el algoritmo continúa con la solución actual. El algoritmo termina cuando el mecanismo de generación no puede encontrar una mejor solución a partir de la solución actual. El algoritmo de búsqueda local en pseudo-código se muestra en la Figura 2.2.

Un concepto importante en el análisis de algoritmos de búsqueda local es el de optimalidad local.

Comienza

INICIALIZA $(i_{inicial})$

$i := i_{inicial}$

Repite

GENERA $(j \in S_i)$

si $f(j) \leq f(i)$ **entonces** $i := j$

hasta $f(j) \geq f(i)$, para toda $j \in S_i$

fin

Figura 2.2: Algoritmo de Búsqueda Local en pseudo-código

Definición 2.7. *Sea (S, f) una instancia de un problema de optimización combinatoria y N una estructura de vecindades, entonces $\bar{i}$ se llama una solución óptima local o simplemente un óptimo local con respecto a N si $\bar{i}$ es mejor que o igual a, todas sus soluciones vecinas con respecto al costo. Específicamente, en el caso de minimización, $\bar{i}$ se llama solución mínima local o simplemente un mínimo local sí*

$$f(\bar{i}) \leq f(j), \text{ para toda } j \in S_{\bar{i}} \tag{2.14}$$

y en el caso de maximización, $\bar{i}$ se llama una solución máxima local o simplemente un máximo local sí

$$f(\bar{i}) \geq f(j), \text{ para toda } j \in S_{\bar{i}} \tag{2.15}$$

Definición 2.8. *Sea (S, f) una instancia de un problema de optimización combinatoria y sea N una estructura de vecindades. Entonces N se llama* ***exacta*** *si para cada $\bar{i} \in S$ que es localmente óptimo con respecto a N, $\bar{i}$ también es globalmente óptimo.*

De aquí se observa que, los algoritmos de búsqueda local terminan con un óptimo local a menos que la estructura de vecindades sea exacta. Generalmente es difícil definir estructuras de vecindades exactas, ya que la definición de una estructura de vecindades exacta frecuentemente involucra hacer una búsqueda exhaustiva en todo el conjunto S.

La calidad del óptimo local obtenido por el algoritmo de búsqueda local, depende fuertemente de la solución inicial y para muchos problemas de optimización combinatoria no se tiene una solución inicial apropiada a la mano. Además la complejidad del problema, en el peor de los casos, no se conoce para muchos problemas.

Entre las ventajas de usar el algoritmo de búsqueda local es que se puede aplicar de manera general a diferentes problemas específicos de optimización combinatoria, sin alterar su estructura, ya que únicamente se requiere de especificar la función de costo y la estructura de vecindades.

Algunas formas de relajar las desventajas del algoritmo de búsqueda local son las siguientes:

1. Ejecutar el algoritmo de búsqueda local para un número grande de diferentes soluciones iniciales. Es claro que de manera asintótica (garantizando que todas las soluciones han sido usadas como inicial) el algoritmo encontrará el óptimo global con probabilidad uno.

2. Introducir estructuras de vecindades complejas, de tal forma que se explore una parte grande del conjunto de soluciones. A menudo encontrar una estructura de este tipo requiere de un conocimiento experto del problema de optimización combinatoria tratado.

3. Aceptar con ciertos límites las transiciones correspondientes a un incremento del valor de la función de costo; note que en el algoritmo de búsqueda local solamente se aceptan transiciones que correspondan a soluciones que decrementen el costo.

2.4 Heurísticas y Metaheurísticas

La palabra heurística proviene del griego *euriskein*, al igual que la famosa palabra eureka y quiere decir encontrar. Esta palabra se volvió muy popular

debido al libro *Cómo resolverlo* (*How to solve it*) de George Pólya, en el cual se proporcionan muchas recetas para tratar de encontrar solución a problemas complejos.

En optimización, la palabra heurística, se utiliza para caracterizar a las técnicas por las cuales se encuentra o se mejora la solución de un problema intratable. Se utilizan algoritmos heurísticos para obtener "buenas soluciones" (soluciones subóptimas, o cercanas a la óptima global) de problemas cuyos algoritmos exáctos de solución no son factibles en tiempos polinomiales como, por ejemplo, el problema del agente viajero.

En computación, es fundamental encontrar algoritmos con buenos tiempos de ejecución y que proporcionen "buenas soluciones", una heurística es un algoritmo que trata de satisfacer al menos uno de estos requerimientos, si no es que ambos. Muchos algoritmos en la inteligencia artificial son heurísticos por naturaleza o usan reglas heurísticas, como ejemplo están los algoritmos que se verán en este libro, los cuales utilizan una amplia variedad de reglas heurísticas.

La palabra metaheurística, combina el prefijo *meta* del griego (que significa "más allá", "a un nivel más alto") con la palabra *heurística*, en otras palabras, se puede decir que *metaheurística* se refiere a una estrategia de nivel superior que guía a otras heurísticas para buscar soluciones factibles para problemas complejos.

En general, las palabras heurística y metaheurística las consideraremos equivalentes cuando se trate de algoritmos de solución para problemas de optimización combinatoria.

Capítulo 3

Problemas de Optimización

En este capítulo se proporcionan varios ejemplos de problemas clásicos que se utilizan para probar la eficiencia, robustes y rapidez computacional de algoritmos. En la primera parte se proporcionan problemas de optimización combinatoria que en su mayoría son del tipo NP difíciles; en la segunda parte se presentan problemas de optimización continua, así como algunas gráficas de éstos.

3.1 Ejemplos de Problemas NP

En esta sección se presentan ejemplos de problemas NP. Algunos de los cuales serán retomados posteriormente.

3.1.1 Problemas P y NP

Un algoritmo es un procedimiento paso a paso para resolver un problema computacional. Para una entrada dada, genera la salida correcta después de un número finito de pasos. La complejidad de tiempo o tiempo de ejecución del algoritmo, expresa el número total de operaciones, como adiciones o comparaciones, como una función del tamaño de la entrada. Se dice que un algoritmo es polinomial o un algoritmo es de tiempo polinomial, si su tiempo de ejecución está acotado por un polinomio con variable en el tamaño de entrada. Se dice que un algoritmo es eficiente si su tiempo de ejecución es polinomial. Los problemas resolubles en tiempo polinomial forman la clase P y están considerados como problemas factibles a resolver, por ejemplo, un

algoritmo muy conocido es el de ordenamiento por selección, que ordena una lista de enteros de la siguiente manera:

1. Encontrar el valor mínimo en la lista.
2. Intercambiarlo con el primer elemento.
3. Repetir los pasos para el resto de la lista.

Para ordenar la lista, (3 1 6 8 2), con el método de selección tenemos los siguientes pasos:

1. (3 1 6 8 2),
2. (1 3 6 8 2),
3. (1 2 6 8 3),
4. (1 2 3 8 6),
5. (1 2 3 6 8).

Podemos determinar el número de comparaciones requeridos por el algoritmo de la siguiente forma. Dada una lista con n elementos, para localizar el valor mínimo, tenemos que revisar todos los n elementos, haciendo $(n-1)$ comparaciones, y después intercambiándolo con la primera posición. Luego, para encontrar el valor mínimo con el resto de la lista, se revisa los $n-1$ elementos restantes, haciendo $n-2$ comparaciones. De esta forma, hacemos en total $(n-1)+(n-2)+...+2+1=\frac{n(n-1)}{2}$ comparaciones.

La manera matemática de describir este comportamiento es con la notación "O grande", donde para dos funciones $f(n)$, $g(n)$ definidas sobre el mismo dominio de enteros positivos, escribimos $f(n)=O(g(n))$ si existe una constante C tal que $f(n)<Cg(n)$. Esto significa que un algoritmo es polinomial si su tiempo de ejecución, con una entrada de tamaño n, es $O(n^k)$ para algún k. En el caso de ordenamiento por selección, podemos tomar $f(n)=\frac{n(n-1)}{2}$ y luego escoger $g(n)=n^2$ y $C=1$. Decimos que ordenamiento por selección tiene complejidad de tiempo de "orden" n^2 o, sencillamente, complejidad $O(n^2)$, y por lo tanto es un algoritmo de tiempo polinomial.

En la Tabla 3.1 vemos el comportamiento aproximado de varias funciones importantes en el análisis de algoritmos. La primera columna indica los valores de la entrada, las demás columnas tienen valores aproximados (en algunos casos) para las funciones que vienen en la primera fila. Las filas 2-6 representan funciones usadas para complejidad polinomial, mientras las últimas dos son de complejidad exponencial y factorial. Aunque los problemas en P con complejidad $O(n^k)$ son considerados como "factibles" a resolver, esto también depende de la constante, la potencia k y el tamaño de entrada n involucrados, como el lector puede apreciar de la Tabla 3.1.

Tamaño n	$log_2 n$	n	$nlog_2 n$	n^2	n^3	2^n	$n!$
10	3.3	10	$(3.3)10$	10^2	10^3	10^3	$(3.6)10^6$
10^2	6.6	10^2	$(6.6)10^2$	10^4	10^6	$(1.3)10^{30}$	$> 10^{157}$
10^3	10	10^3	$(1.0)10^3$	10^6	10^9	$> 10^{300}$	
10^4	13	10^4	$(1.3)10^4$	10^8	10^{12}		
10^5	17	10^5	$(1.7)10^5$	10^{10}	10^{15}		
10^6	20	10^6	$(2.0)10^6$	10^{12}	10^{18}		

Tabla 3.1: Valores aproximados de algunas funciones.

Otra clase de problemas, como el agente viajero, son conocidos como problemas NP, donde NP significa "polinomial no-determinista". Para estos problemas, no se conocen algoritmos con tiempo polinomial que los resuelvan, y generalmente se cree que tampoco existen, aunque la pregunta: ¿P = NP? es decir, si las dos clases de problemas coinciden, es considerada como una de las más importantes en la ciencias computacionales teóricas.

También hay una subclase importante de problemas NP conocida como NP-Completa, de los cuales, si se puede demostrar que uno de ellos es resoluble en tiempo polinomial, entonces también cualquier problema en NP-Completa sería resoluble en tiempo polinomial. Además la clase NP-Completa tiene interés práctico porque contiene muchos problemas comunes, como el Problema del Agente Viajero. El lector puede encontrar más información en [103].

3.1.2 Resolviendo Problemas NP.

Por lo que hemos dicho en la sección 3.1, todavía no podemos resolver los problemas NP-completos con algoritmos eficientes, es decir, de tiempo poli-

nomial; sin embargo, estos problemas ocurren con frecuencia. ¿Qué podemos hacer?

En parte, depende si quisieramos una solución exacta o aproximada al problema. Por ejemplo, si el tamaño de entrada del problema no es grande, puede ser factible encontrar la solución exacta con algún método directo. En el caso del Problema del Agente Viajero, la solución más directa es la de intentar todas las permutaciones de las n ciudades y encontrar la más barata, dando una complejidad de $O(n!)$. Desafortunadamente, con menos de 100 ciudades vemos, en la Tabla 3.1 que rápidamente encontramos problemas, pero puede haber otros enfoques especializados, por ejemplo con programación dinámica, que reduce la complejidad y permite resolver exactamente para un número mayor de ciudades; en el problema del agente viajero, se ha ido resolviendo desde 49 ciudades en 1954, al record actual de 85,900 ciudades en 2006 [11], el último tomando 136 años de tiempo CPU.

Otra posibilidad es utilizar un método heurístico, que no garantiza encontrar la solución exacta, pero puede encontrar "buenas soluciones" en tiempos razonables. Los temas tratados en los siguientes capítulos del libro introducen algunos de ellos y se aplican a diversos problemas.

A continuación, se presentan algunos ejemplos de problemas NP. El lector puede encontrar una colección amplia de éstos en [30].

3.1.3 Problema de Asignación Cuadrática

Una formulación matemática de esta problemática fue inicialmente planteada por Koopmans y Beckmann ([68]) y consiste en minimizar:

$$f(\pi) = \sum_{i=1}^{n} \sum_{k=1}^{n} a_{ik} b_{\pi(i)\pi(k)}$$

donde π es una permutación del conjunto $N = \{1, ...n\}$ y a_{ik}, b_{ik} para i, $k = 1,...,n$, son números reales. Equivalentemente, se desea encontrar una permutación π del conjunto N que minimice el valor de $f(\pi)$.

Una forma sencilla de interpretar el problema planteado es suponer que existen n sitios disponibles y se van a construir n edificios en estos lugares. Sea a_{ik} la distancia entre el sitio i y el sitio k y suponga que las cantidades b_{jl} denoten el número de personas por semana que viajarán entre el edificio j y l.

El problema es asignar los sitios de construcción de manera que la distancia recorrida se minimice. Cada asignación puede describirse matemáticamente como una permutación π del conjunto $N = \{1, 2, ..., n\}$, mientras que el producto $a_{ik}b_{\pi(i)\pi(k)}$ describe la distancia semanal recorrida por la gente que viaja entre los edificios $j = \pi(i)$ y $l = \pi(k)$, si la construcción j se realiza en el sitio i y la construcción l se realiza en el sitio k.

Considere un ejemplo numérico donde se desea localizar cuatro edificios A, B, C y D en cuatro sitios a, b, c, d. Suponga que las distancias (en metros) entre los cuatro edificios están dadas por la matriz de distancias (a_{ik}):

$$(a_{ik}) = \begin{bmatrix} 0 & 340 & 320 & 400 \\ 340 & 0 & 360 & 200 \\ 320 & 360 & 0 & 180 \\ 400 & 200 & 180 & 0 \end{bmatrix}$$

El edificio A es la biblioteca, el edificio B son aulas y cubículos, el edificio C son oficinas administrativas y el edificio D son dormitorios. Las "conexiones" entre estos edificios se describen por medio de la matriz de conexión (b_{jl}), donde b_{jl} es una medida para el número de personas y la frecuencia con la que viajan del edificio j al edificio l:

$$(b_{jl}) = \begin{bmatrix} 0 & 80 & 40 & 30 \\ 80 & 0 & 30 & 20 \\ 40 & 30 & 0 & 10 \\ 30 & 20 & 10 & 0 \end{bmatrix}$$

Observe que si se construye el edificio A en el sitio a, el edificio B en el sitio b, el edificio C en el sitio c y el edificio D en el sitio d, entonces se obtendría el siguiente valor de la función objetivo:

$$z = \sum_{i=1}^{n} \sum_{k=1}^{n} a_{ik}b_{ik} = 137,200.$$

que corresponde a la permutación $\pi(i) = i$ para $i = 1, 2, 3, 4$. Por otro lado, suponga que A se construye en el sitio d, B en b, C en c y D en a, se tiene la permutación $\pi(1) = 4$, $\pi(2) = 2$, $\pi(3) = 3$, $\pi(4) = 1$ y el valor de la función objetivo es ahora.

$$z = \sum_{i=1}^{n} \sum_{k=1}^{n} a_{ik} b_{\pi(i)\pi(k)} = 112,000.$$

Otra interpretación del problema planteado consiste en suponer que n módulos se han colocado en un tablero y se han conectado por medio de "alambres". Los módulos deben colocarse de manera que la longitud total del alambre que los conecta sea mínima.

Denote la distancia entre las posiciones i y k en el tablero por a_{ik}. Sea b_{jl} el número de conexiones entre el módulo j y el módulo l. Entonces, una permutación π del conjunto $N = \{1, 2, ..., n\}$ asigna una posición en el tablero a todo módulo. Si j se sitúa en la posición i y l está situado en la posición k, el producto $a_{ik} b_{\pi(i)\pi(k)}$ es la longitud del alambre entre los módulos $j = \pi(i)$ y $l = \pi(k)$. Por lo tanto la longitud total del alambrado es:

$$\sum_{i=1}^{n} \sum_{k=1}^{n} a_{ik} b_{\pi(i)\pi(k)},$$

y el problema se reduce a:

$$\min_{\pi} z = \sum_{i=1}^{n} \sum_{k=1}^{n} a_{ik} b_{\pi(i)\pi(k)}$$

El problema anterior, se conoce comúnmente en la literatura como *el Problema de Asignación Cuadrática* (por sus siglas en inglés, QAP). Una formulación similar del QAP es aplicable para el diseño de paneles de control y teclados de máquinas. Existen muchas otras aplicaciones del QAP por ejemplo:

1. Problemas de ubicación de edificios, planeación de hospitales.

2. En deportes, se aplica en problemas de calendarización.

3. En sistemas de información, un orden óptimo para interrelacionar datos en una cinta magnética.

4. En química, para analizar reacciones químicas para componentes orgánicas.

5. En balancear turbinas para autos de carreras.

6. En arqueología, modelos para ordenar datos arqueológicos.

3.1.4 Problema de Conjunto Independiente

Considere una gráfica $G = (V, E)$ no dirigida, donde V es el conjunto de vértices y E es el conjunto de aristas. Un subconjunto I de V se dice que es un *conjunto independiente* de vértices, también conocido como *conjunto estable*, de G si no existen dos vértices en I que sean adyacentes; es decir, no existen dos vértices que estén ligados por una arista. Para la Figura 3.1, por ejemplo, los conjuntos de vértices {1,5,7}, {2,6}, {2,4,6}, son conjuntos independientes.

Un conjunto independiente se llama *maximal* si no existe otro conjunto que sea independiente en el que esté contenido propiamente. Esto es, un conjunto independiente I es maximal si para todo conjunto H tal que $I \subset H$ con $I \neq H$, se tiene que H no es independiente. Por ejemplo, para la Figura 3.1, los conjuntos {1,5,7}, {2,4,6}, {3,6} y {1,8}, son conjuntos independientes maximales y los conjuntos {1}, {2,6}, {1,5} no son maximales.

En general, existe más de un conjunto independiente maximal para una gráfica dada. También, como se vio en los ejemplos anteriores, el número de vértices en los conjuntos independientes maximales no es necesariamente el mismo.

Si Q es la familia de conjuntos independientes asociados a una gráfica G, entonces el número:

$$\alpha\,|G| = \max_{I \in Q} |I|$$

se llama el *número independiente* o de *estabilidad* de una gráfica G, y el conjunto I^* para el que se obtiene este número se llama un *conjunto independiente máximo*. El Problema de Conjunto Independiente consiste en encontrar dentro de todos los conjuntos independientes I en Q, un conjunto independiente I^* cuya cardinalidad sea máxima; es decir, que tenga el número máximo de elementos. En la Figura 3.1, la familia de conjuntos maximales está compuesta por:

$$\{\{1,5,7\},\{1,6\},\{1,8\},\{2,4,6\},\{2,4,8\},\{3,5\},\{3,6\},\{3,8\},\{4,5\}\}$$

el más grande de estos conjuntos tiene 3 elementos y por lo tanto $\alpha\,|G| = 3$. Cualquier conjunto de la familia maximal con 3 elementos es un conjunto independiente máximo, por lo tanto, los conjuntos independientes máximos para la gráfica de la Figura 3.1 son: {1,5,7}, {2,4,6} y {2,4,8}.

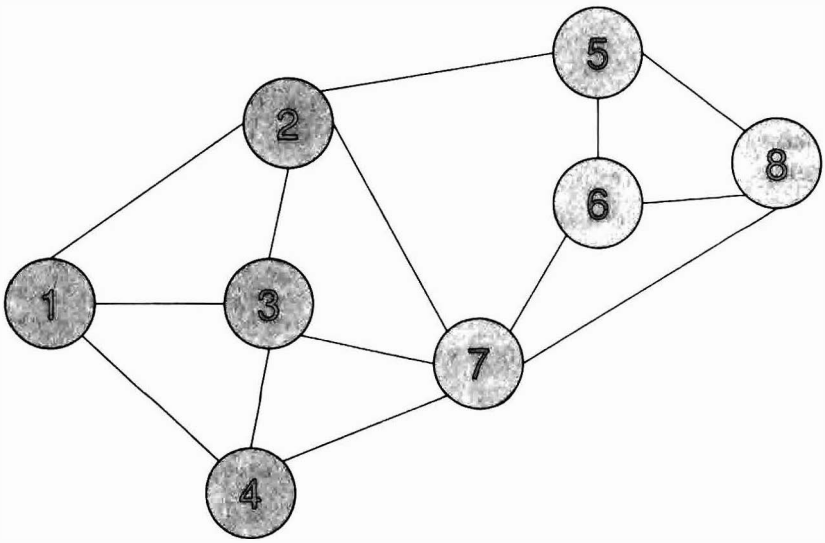


Figura 3.1: Gráfica con 8 vértices y 14 aristas.

3.1.5 Problema de Coloración Mínima

Considere una gráfica $G = (V, E)$ no dirigida, donde V es el conjunto de vértices y E es el conjunto de aristas. Se define una *coloración* de G como una aplicación $C : V \rightarrow \{1, 2, ..., n\}$ que identifica a $C(i)$ como el color del vértice i, de forma tal que dos vértices adyacentes no tengan el mismo color, es decir, $\{i, j\} \in E \Rightarrow C(i) \neq C(j)$.

Una coloración de una gráfica G con k colores define una partición del conjunto de vértices en subconjuntos $V_1, V_2, \ldots, V_k$, donde V_j denota los vértices que tienen asignado el color j. Claramente, cada V_j es un conjunto independiente, al cual se le llama *clase de color* j o *clase cromática* j. Se dice que una *k-coloración* de G es una coloración que no utiliza más de k colores, y una gráfica es k-coloreable si admite una k-coloración. Al mínimo valor k tal

que G es k-coloreable se le llama *número cromático* de la gráfica y se denota por $\chi(G)$.

En una gráfica G, el Problema de Coloración Mínima busca una coloración de G que no utilice más de $\chi(G)$ colores.

Por ejemplo la Figura 3.1 es k-coloreable para $k = 3,4,5,\ldots$ si seleccionamos $k = 3$ se puede colorear los vértices 1, 5 y 7 con el color 1, los vértices 2, 4 y 6 con el color 2 y los vértices 3 y 8 con el color 3. Es decir, $V_1 = \{1,5,7\}, V_2 = \{2,4,6\}$ y $V_3 = \{3,8\}$. También se puede observar que $\chi(G) = 3$, ya que para colorear, por ejemplo, los vértices 1, 2 y 3 se requieren colores distintos.

3.1.6 Problema de Empaquetamiento

El *Problema de Empaquetamiento* (Bin Packing Problem) consiste en acomodar el mayor número de objetos (cada uno con determinado peso o valor) en el menor número de contenedores (cajas, huecos, camiones, etc.). En el problema de empaquetamiento se tienen n objetos $a_1, a_2, \ldots, a_n$, los cuales tienen un peso $w(a_i) \in (0,1]$ que serán empaquetados en un número mínimo de contenedores $B_1, B_2, \ldots, B_m$ tal que $\sum_{a_i \in B_j} w(a_i) \leq 1, \quad 1 \leq j \leq m$.

Por ejemplo, si se tienen 5 objetos a_1, a_2, a_3, a_4, a_5, con pesos $w(a_1) = 0.8, w(a_2) = 0.6, w(a_3) = 0.5, w(a_4) = 0.2, w(a_5) = 0.3$. Para este ejemplo, el mínimo número de contenedores que se requieren es 3 y una posible solución es: $B_1 = \{a_1\}, B_2 = \{a_2, a_5\}, B_3 = \{a_3, a_4\}$.

3.1.7 Problema de Cubrimiento de Vértices

Dada una gráfica no dirigida $G = (V, E)$, donde V es el conjunto de vértices y E es el conjunto de aristas, un *cubrimiento por vértices* es un conjunto C de vértices tal que cada arista de G tiene al menos uno de sus vértices adyacentes en C. El *Problema de Cubrimiento de Vértices* (*The Vertex Cover Problem*) consiste en encontrar un cubrimiento por vértices más pequeño.

Por ejemplo, para la Figura 3.1, un cubrimiento por vértices está dado por los conjuntos: $C_1 = \{1,2,3,6,7,8\}$, $C_2 = \{1,3,5,6,7\}$, $C_3 = \{2,3,4,6,8\}$, $C_4 = \{1,2,3,4,5,6,7,8\}$. Los conjuntos C_2 y C_3 son dos posibles soluciones al problema. No existe ningún cubrimiento que tenga 4 o menos elementos.

3.1.8 Problema del Agente Viajero

El Problema del Agente Viajero consiste, como se vió anteriormente, en dado un entero $n > 0$ y la distancia entre cada par de las n ciudades se da por medio de la matriz (d_{ij}) de dimensión $n \times n$, donde d_{ij} es un entero mayor o igual a cero. Un recorrido es una trayectoria que visita todas las ciudades exactamente una vez. El problema consiste en encontrar un recorrido con longitud total mínima.

Una permutación π cíclica representa un recorrido, si se interpreta $\pi(j)$ como la ciudad después de la ciudad $j, j = 1, 2, \ldots, n$. Así, el costo del recorrido es:

$$\sum_{j=1}^{n} d_{j\pi(j)}$$

3.1.9 Problema de Clan Máximo.

Considere una gráfica $G = (V, E)$ no dirigida, donde V es el conjunto de vértices y E es el conjunto de aristas. Un subconjunto $S \subset V$se dice que es un *clan* si para cada par de vértices v_1, v_2 en S con $v_1 \neq v_2$ existe la arista (v_1, v_2) en E. El *Problema de Clan Máximo* (*Maximum Clique*) consiste en encontrar un subconjunto S de vértices tal que para cada par de vértices en S existe una arista en G y tal que $|S|$ sea máximo.

En la Figura 3.1, los conjuntos $S_1 = \{1, 2, 3\}, S_2 = \{5, 6, 8\}$ y $S_3 = \{6, 8\}$ son clanes de tamaño 3, 3 y 2 respectivamente. Los conjuntos S_1 y S_2 son dos posibles soluciones al problema. No existe ningún clan de cardinalidad mayor.

3.1.10 Problema de Corte Máximo.

Considere una gráfica $G = (V, E)$ no dirigida, donde V es el conjunto de vértices y E es el conjunto de aristas. El *Problema de Corte Máximo* (Maximum Cut) se refiere a encontrar un subconjunto $C \subset V$que maximiza el número de aristas que tienen un vértice en C y el otro fuera de C.

En la Figura 3.1, el conjunto $C_1 = \{3, 5, 6, 8\}$ hace que el conjunto de aristas $A_1 = \{(1, 3), (2, 5), (2, 3), (3, 4), (3, 7), (6, 7), (7, 8)\}$. Tengan un vértice en C_1 y el otro fuera de C_1.

3.1.11 Problema de Satisfacibilidad

Una variable booleana x es una variable que toma solamente valores de cierto o falso. Las variables booleanas pueden combinarse usando los conectores lógicos 'o' (denotado aquí por $+$), 'y' (denotado como multiplicación), y 'no' (denotado por $\neg$) para formar una expresión booleana, se hace lo mismo que con variables reales, se combinan con operaciones algebraicas. Por ejemplo,

$$\neg x_3 \cdot (x_1 + \neg x_2 + x_3)$$

es una fórmula booleana. Dados los valores de cierto o falso para cada variable, se puede evaluar la fórmula booleana, justo como una expresión algebraica. Por ejemplo, para la fórmula booleana anterior tomando $x_1 =$ cierto, $x_2 =$ cierto, y $x_3 =$ falso, la expresión da el valor cierto.

La fórmula anterior puede hacerse cierta para algunas asignaciones. Tal fórmula booleana se dice que es *satisfacible*. No todas las fórmulas son satisfacibles; hay algunas que no se pueden hacer ciertas para ninguna asignación, esencialmente porque tienen alguna "contradicción". Por ejemplo, considere

$$(x_1 + x_2 + x_3) \cdot (x_1 + \neg x_2) \cdot (x_2 + \neg x_3) \cdot (x_3 + \neg x_1) \cdot (\neg x_1 + \neg x_2 + \neg x_3)$$

Para que la fórmula anterior sea cierta, todas las subfórmulas dentro de paréntesis (llamadas cláusulas) que contienen literales (esto es, variables o negaciones) deben ser ciertas. La primera cláusula dice que al menos una de las variables debe ser cierta. Las siguientes tres cláusulas forzan a todas las variables a tener el mismo valor y como al menos una debe ser cierta (de la primera cláusula), entonces todas deben ser ciertas. A pesar de esto, la última cláusula requiere que no todas sean ciertas, y así la fórmula contiene una contradicción, por lo tanto es no satisfacible. El problema de satisfacibilidad se puede enunciar como:

Dadas m cláusulas $C_1, ..., C_m$ que contienen las variables $x_1, ..., x_n$, ¿Es la fórmula $C_1 \cdot C_2 \cdots C_m$ satisfacible?

3.1.12 Problema de la Mochila

Dado un conjunto $U = \{a_1, a_2, \ldots, a_n\}$, de n elementos con pesos p_i, $i = 1, 2, \ldots, n$ y un parámetro $B > 0$. El objetivo es encontrar un subconjunto $U' \subseteq U$ tal que $\sum_{j \in U'} p_j \leq B$ sea máxima. A este problema se le llama el *Problema de la Mochila* (*Knapsack Problem*)

3.1.13 Cuadrados Mágicos

Dada una matriz de 4×4 como:

$$\begin{bmatrix} 16 & 3 & 2 & 13 \\ 5 & 10 & 11 & 8 \\ 9 & 6 & 7 & 12 \\ 4 & 15 & 14 & 1 \end{bmatrix}$$

Esta matriz no es un arreglo al azar. Todos los enteros del 1 al 16 se usan, la suma de cada renglón, columna y las dos diagonales son iguales a 34. Este tipo de matrices se conocen como cuadrados mágicos. Hay cuadrados mágicos de cualquier tamaño, excepto de 2×2. Claramente la suma común de los renglones columnas y diagonales está en función de n. Para $n = 3$, la suma es 15.

Para valores de $n = 3$, 4 y 5 aquí hay unos cuadrados mágicos:

$$\begin{bmatrix} 8 & 1 & 6 \\ 3 & 5 & 7 \\ 4 & 9 & 2 \end{bmatrix} \qquad \begin{bmatrix} 16 & 3 & 2 & 13 \\ 5 & 10 & 11 & 8 \\ 9 & 6 & 7 & 12 \\ 4 & 15 & 14 & 1 \end{bmatrix} \qquad \begin{bmatrix} 17 & 24 & 1 & 8 & 15 \\ 23 & 5 & 7 & 14 & 16 \\ 4 & 6 & 13 & 20 & 22 \\ 10 & 12 & 19 & 21 & 3 \\ 11 & 18 & 25 & 2 & 9 \end{bmatrix}$$

Así, el problema de *Cuadrados Mágicos* (*Magic Squares*) es a partir de un valor entero $n > 2$ se debe construir una matriz cuadrada de $n \times n$ con entradas del 1 al $n \times n$ tal que la suma de cada renglón, columna y las dos diagonales son iguales a una constante que depende del valor n.

3.1.14 El Problema del Sudoku

El Sudoku es un pasatiempo muy popular a nivel mundial. El nombre, Sudoku, viene de japones donde 'Su' significa número y 'Doku' significa individualmente.

En el caso general, el objetivo del Sudoku es llenar una cuadrícula de $n^2 \times n^2$ celdas, dividida en n^2 subcuadrículas de $n \times n$ en donde se deben colocar los enteros del 1 a n^2 de acuerdo a los siguientes tres reglas:

1. Las celdas de cada uno de los renglones deben contener los enteros de 1 a n^2 exactamente una vez.
2. Las celdas de cada una de las columnas deben contener los enteros de 1 a n^2 exactamente una vez.
3. Cada una de las subcuadrículas deben contener los enteros de 1 a n^2 exactamente una vez.

Al entero n se le denomina el orden del Sudoku.

En la Figura 3.2 se da un ejemplo de un Sudoku de orden 3, donde se puede observar que existen algunas celdas prellenadas con los enteros del 1 al 3^2 y el jugador debe completar la cuadrícula siguiendo las tres reglas descritas anteriormente. En la Figura 3.3 se presenta una solución válida de la Figura 3.2.

1			7	3				
						2	5	
8								
				4				1
		5			6			
	2							
4							3	
					2	4		
	1		5					

Figura 3.2: Sudoku.

En 2002, [115] mostró que el Sudoku es un problema NP-Completo. La prueba usa una reducción simple para el problema de cuadrados latinos, el cual ha sido probado que es NP-Completo.

1	5	2	**7**	**3**	8	9	6	4
7	3	9	4	6	1	**2**	**5**	8
8	6	4	2	5	9	1	7	3
6	8	7	9	**4**	5	3	2	**1**
3	4	**5**	1	2	**6**	8	9	7
9	**2**	1	8	7	3	6	4	5
4	9	8	6	1	7	5	**3**	2
5	7	6	3	8	**2**	**4**	1	9
2	**1**	3	**5**	9	4	7	8	6

Figura 3.3: Solución válida al Sudoku de la Figura 3.2.

3.1.15 Problema de Asignación Tridimensional Axial

El *Problema de Asignación Tridimensional Axial* (*Three Dimensional Axial Assigment Problem*) consiste en: dados tres conjuntos I, J, K, todos de igual cardinalidad n, y un costo c_{ijk} asociado a cada terna ordenada (i, j, k) perteneciente a $I \times J \times K$, hallar n ternas disjuntas con menor suma de costos. Este problema es NP-Duro. El problema se representa a través del siguiente modelo matemático:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} c_{ijk} x_{ijk}$$

Sujeto a :

$$\begin{aligned} \sum_{j=1}^{n} \sum_{k=1}^{n} c_{ijk} x_{ijk} &= 1, \ i = 1, ..., n \\ \sum_{i=1}^{n} \sum_{k=1}^{n} c_{ijk} x_{ijk} &= 1, \ j = 1, ..., n \\ \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ijk} x_{ijk} &= 1, \ k = 1, ..., n \\ x_{ijk} &\in \{0, 1\}, \quad i, j, k = 1, ..., n \end{aligned}$$

3.1.16 Problema de Aprendizaje en Perceptrones Multicapa

Los perceptrones multicapa (PMC) ocupan un lugar importante en el área del aprendizaje maquinal, donde han sido aplicados, por ejemplo, a distintas tareas de clasificación, como el reconocimiento de caracteres manuscritos. Los PMC están biológicamente motivados por la organización del cerebro humano, donde intentan imitar el procesamiento en paralelo que realiza el cerebro.

Podemos definir un PMC en términos de una colección de capas ordenadas, empezando con una capa de entrada y terminando con una capa de salida; las capas intermedias se llaman "capas ocultas", debido a que las entradas a la red son externas, así como las salidas que el PMC produce. Cada capa contiene un cierto número de nodos (a veces llamados unidades de procesamiento o neuronas), y cada nodo tiene conexiones con cada nodo en la siguiente capa. La Figura 3.4 muestra un PMC con una capa oculta con tres nodos, dos nodos en la capa de entrada y dos en la capa de salida. En la Figura 3.4, las flechas representan las conexiones entre los nodos.

En cada conexión hay un valor numérico asociado, llamado peso. El PMC recibe un valor numérico externo en cada nodo de entrada, y estos valores son pasados directamente a los nodos en la siguiente capa, sin ningún procesamiento. En los nodos de las siguientes capas, se calcula un valor de salida dado por:

$$y = g\left(\sum_{i=1}^{k} a_i u_i + b\right)$$

donde $u_1, \ldots, u_k$ son las salidas de los nodos en la capa anterior, los coeficientes a_i son los pesos, b es un umbral, y la función g se llama función de activación. Es común usar la misma función de activación en los nodos de las capas ocultas y de salida, y con frecuencia está tomada como un sigmoide definida por: $g : \mathbb{R} \longrightarrow [0,1]$, $g(u) = \frac{1}{1+\exp^{-u}}$

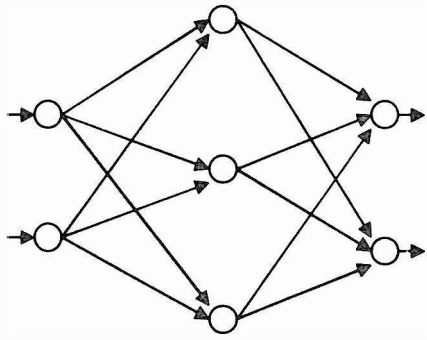


Figura 3.4: Un perceptrón multicapa.

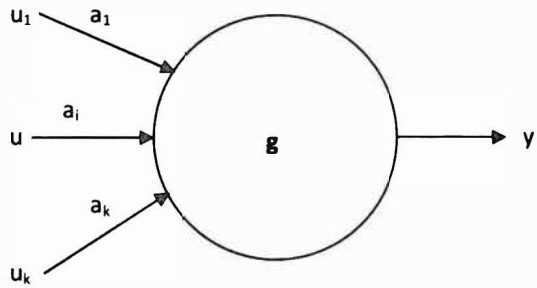


Figura 3.5: Procesamiento realizado en un nodo en una capa oculta o de salida.

La Figura 3.5 ilustra este procesamiento realizado por un nodo. Este proceso procede por capas, de tal forma que, los valores de las salidas en una capa son calculadas y después pasadas a la siguiente capa. Finalmente los valores de salida de los nodos en la capa de salida se encuentran de acuerdo a los valores de entrada y este vector de valores representa la salida del PMC. Conforme a esta descripción, un PMC con n nodos en la capa de entrada y m en la capa de salida define una función $F : \mathbb{R}^n \longrightarrow \mathbb{R}^m$.

Dado un conjunto de vectores $X = \{(x_1, d_1), \ldots, (x_N, d_N)\}$ con x_i en $\mathbb{R}^n$ y d_i en $\mathbb{R}^m$, para $i = 1, \ldots, N$, podemos preguntar: ¿Existen pesos y umbrales tales que: $F(x_i) = d_i$, para cada $i = 1, \ldots, N$? Éste se llama el *Problema de Entrenamiento o Aprendizaje* (a veces llamado en inglés el *Loading Problem*) de un PMC, y a X se le llama un *conjunto de entrenamiento.*

Se han desarrollado varios algoritmos, principalmente el algoritmo de retropropagación (en inglés, *back-propagation de Rumelhart* [96]), para resolver el problema de aprendizaje. Este algoritmo está basado en el método de gradiente descendente para minimizar la función objetivo, E_X, formada por el error cuadrático con respecto al conjunto de entrenamiento X, que viena dada por:

$$E_X(w) = \sum_{k=1}^{N} ||y_k - d_k||^2$$

donde y_k son los vectores de salida del PMC para cada vector de entrenamiento, x_k, w son los pesos y umbrales, y $|| \cdot ||$ es la norma usual euclideana en $\mathbb{R}^m$.

Hay que tomar en cuenta que en estudios empíricos se ha visto que el tiempo de aprendizaje con el algoritmo de retropropagación parece escalar ex-

ponencialmente con el tamaño de los conjuntos de entrenamiento, sugiriendo que hay un problema inherente con el entrenamiento de PMC. Judd [62] estudió el problema de aprendizaje y mostró que bajo condiciones "razonables" es NP-Completo. Desde entonces se han encontrado varios resultados similares, de los cuales mencionamos el siguiente que es sorprendente, dado por Sima [98], que no requiere una clasificación exacta de todos los datos de entrenamiento, sino más bien una que puede tener pequeños errores en las salidas y muestra que el problema de entrenamiento es NP-Duro aún para el caso de un sólo nodo. Más específicamente, considerar un PMC con n nodos de entrada, un nodo de salida, y sin capas ocultas, y con una función de activación sigmoide (en realidad, Sima consideró funciones aún más generales). Dado un conjunto de entrenamiento X y un número positivo $\epsilon > 0$ ¿Existe un vector w de los pesos y del umbral tal que $E_X(w) \leq \epsilon$? Sima muestra que éste es un problema NP-Duro.

3.2 Problemas Continuos

Para la comparación de diferentes algoritmos diseñados para resolver problemas de optimización continua, a lo largo de los años se ha propuesto una colección de funciones de prueba. Las funciones están escogidas para reflejar diferentes facetas que podrían resultar difíciles para un algoritmo. Estas propiedades de una función son: unimodalidad, multimodalidad, dimensionalidad, continuidad, diferenciabilidad, convexidad, el número de óptimos locales y su distribución, el número de óptimos globales y su distribución.

A continuación, definimos algunas de las funciones que se han utilizadas ([83], [80]), formulando todos los problemas para minimización.

3.2.1 La función de Ackley

La función de Ackley es una función multimodal. Está definida por:

$$f(x_1, x_2, \ldots, x_n) = -20\exp^{-0.2\sqrt{\sum_{i=1}^{N} x_i^2/N}} - \exp^{(\sum_{i=1}^{N}\cos(2\pi x_i))/N} + 20 + \exp^1$$

$$-32.768 \leq x_i \leq 32.768, \quad i = 1, 2, \ldots, N$$

Tiene un mínimo global de $f(x_1, x_2, \ldots, x_N) = 0$ en $x_i = 0, i = 1, 2, \ldots N$.

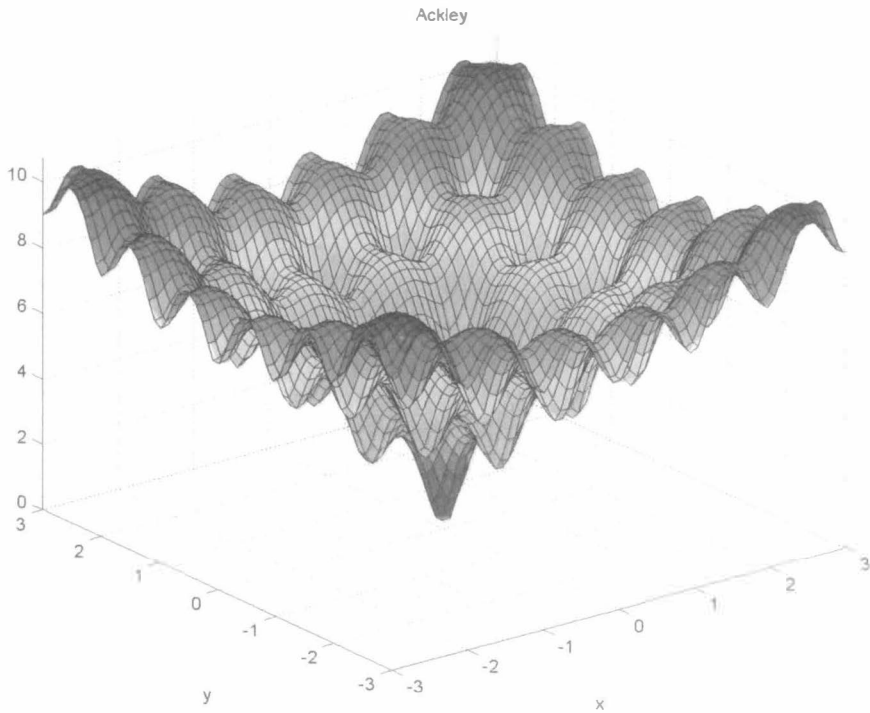


Figura 3.6: La función de Ackley en dos dimensiones.

3.2.2 La función RCOS de Branin

La función rcos de Branin tiene un mínimo global en tres diferentes puntos. Está definida por:

$$f(x,y) = a(y - bx^2 + cx - d)^2 + e(1-f)\cos(x) + e, \quad -5 \leq x \leq 10, \quad 0 \leq y \leq 15$$

donde usualmente se toma:

$$a = 1, \quad b = 5.1/(4\pi^2), \quad c = 5/\pi, \quad d = 6, \quad e = 10, \quad f = 1/(8\pi).$$

Tiene el mínimo global de $f(x,y)$ =0.397887 en los puntos: $x = -\pi$, $y = 12.275$, en $x = \pi$, $y = 2.275$, y en $x = 9.42478$, $y = 2.475$.

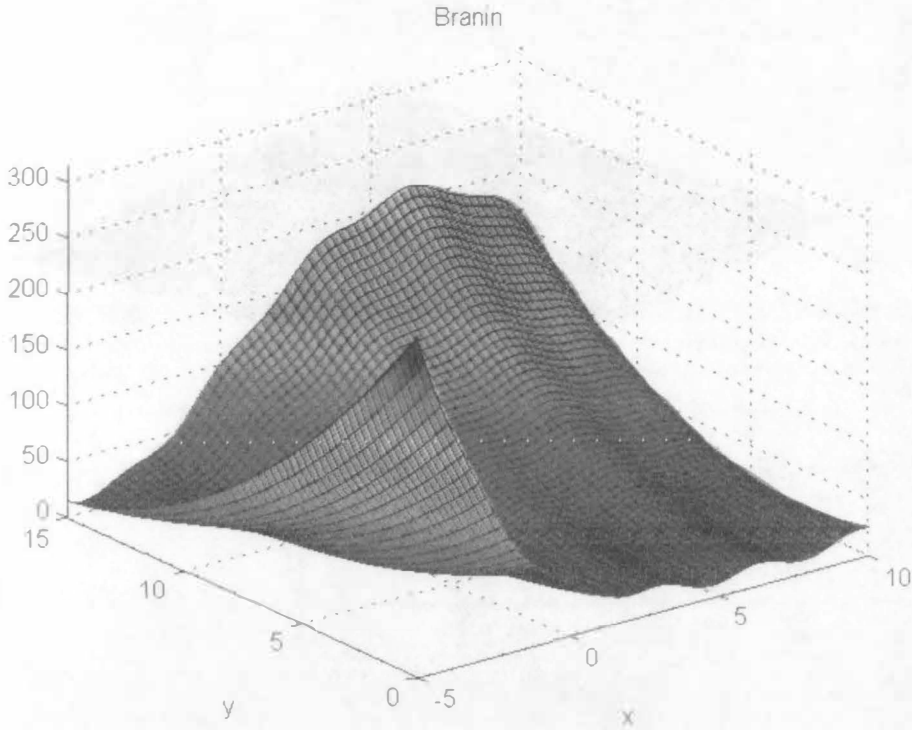


Figura 3.7: La función rcos de Branin.

3.2.3 La función de Easom

La función de Easom es una función unimodal, donde el mínimo global se encuentra en un área pequeño comparado con el resto del dominio.

$$f(x,y) = -\cos(x)\cos(y)\exp^{-(x-\pi)^2-(y-\pi)^2},$$

$$-100 \leq x, y \leq 100.$$

Tiene un mínimo global de $f(x,y) = -1$ en $x = y = \pi$.

3.2.4 La función de la esfera

En el contexto de optimización, esta función también esta conocida como la primera de De Jong. Es una función continua, convexa, y unimodal definida

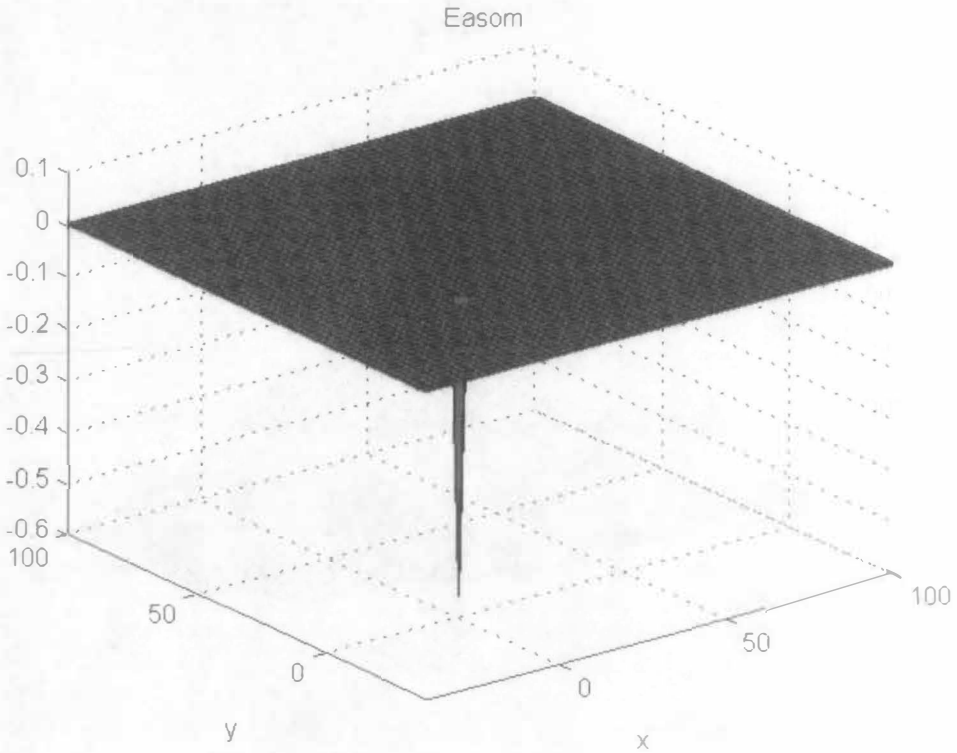


Figura 3.8: La función Easom.

por:

$$f(x_1, x_2, \ldots, x_N) = \sum_{i=1}^{N} x_i^2,$$

$$-5.12 \leq x_i \leq 5.12, \quad i = 1, 2, \ldots, N.$$

Tiene un mínimo global de $f(x_1, x_2, \ldots, x_N) = 0$ en $x_i = 0, \ i = 1, \ldots, N$.

3.2.5 La función de Goldstein-Price

La función de Goldstein-Price está definida en dos dimensiones por:

$$f(x, y) = (1 + (x + y + 1)^2(19 - 14x + 3x^2 - 14y + 6xy + 3y^2))$$
$$(30 + (2x - 3y)^2(18 - 32x + 12x^2 + 48y - 36xy + 27y^2)),$$

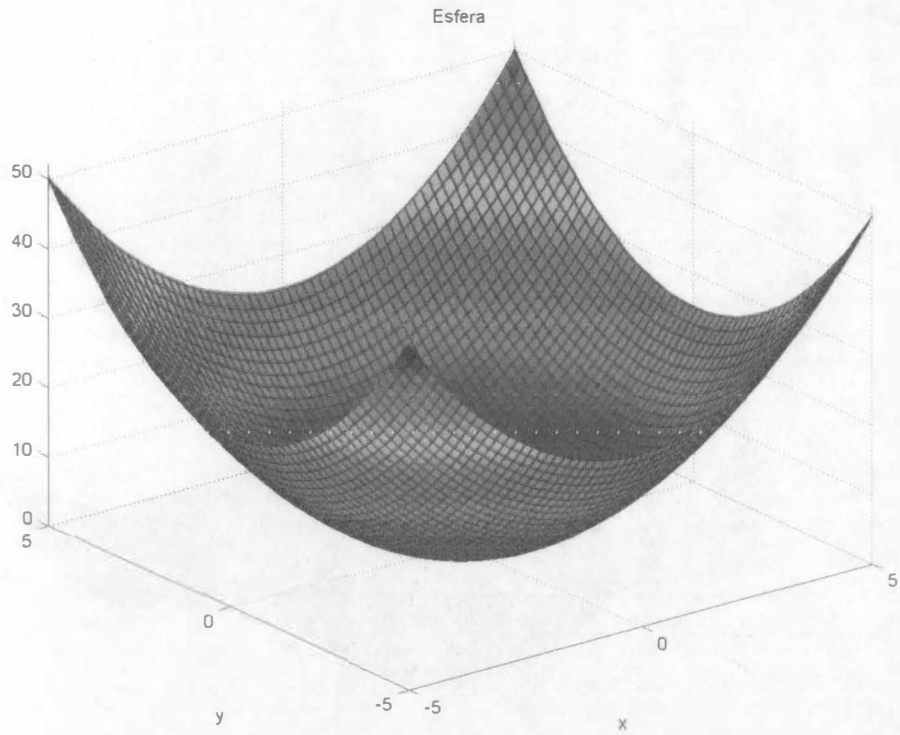


Figura 3.9: La función de De Jong en dos dimensiones.

para valores $-2 \leq x, y \leq 2$. Tiene un mínimo global de $f(x, y) = 3$ en $x = 0, y = -1$.

3.2.6 La función de Griewangk

La función de Griewangk es multimodal con muchas mínimas locales que están distribuidas de manera regular. Está definida por:

$$f(x_1, x_2, \ldots, x_N) = \frac{1}{4000} \sum_{i=1}^{N} x_i^2 - \prod_{i=1}^{N} \cos(\frac{x_i}{\sqrt{i}}) + 1,$$

$$-600 \leq x_i \leq 600, i = 1, 2, \ldots, N,$$

Tiene un mínimo global de $f(x_1, x_2, \ldots, x_N) = 0$ en $x_i = 0$, $i = 1, ..., N$.

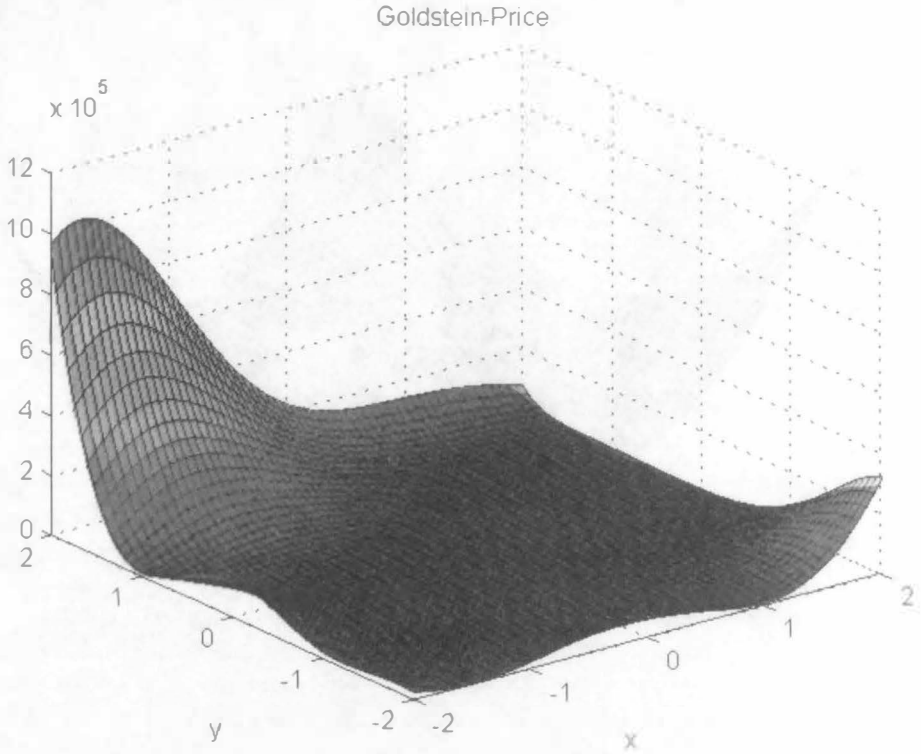


Figura 3.10: La función Goldstein-Price.

Como se puede apreciar en la figura 3.11, conforme vamos acercando al mínimo global, la función cambia de lo que aparece una función convexa, a una función con muchas mínimos locales.

3.2.7 La función de Langermann

La función de Langermann es una función multimodal, donde sus mínimos locales están distribuidos de manera no pareja. Está definida por:

$$f(x_1, x_2, \ldots, x_N) = \sum_{i=1}^{N} c_i \exp\left[-\frac{1}{\pi}\sum_{j=1}^{N}(x_j - a_{ij})^2\right] \cos\left[\pi \sum_{j=1}^{N}(x_j - a_{ij})^2\right]$$

$$2 \leq m \leq 10, \quad 0 \leq x_i \leq 10, \quad i = 1, 2, \ldots, N.$$

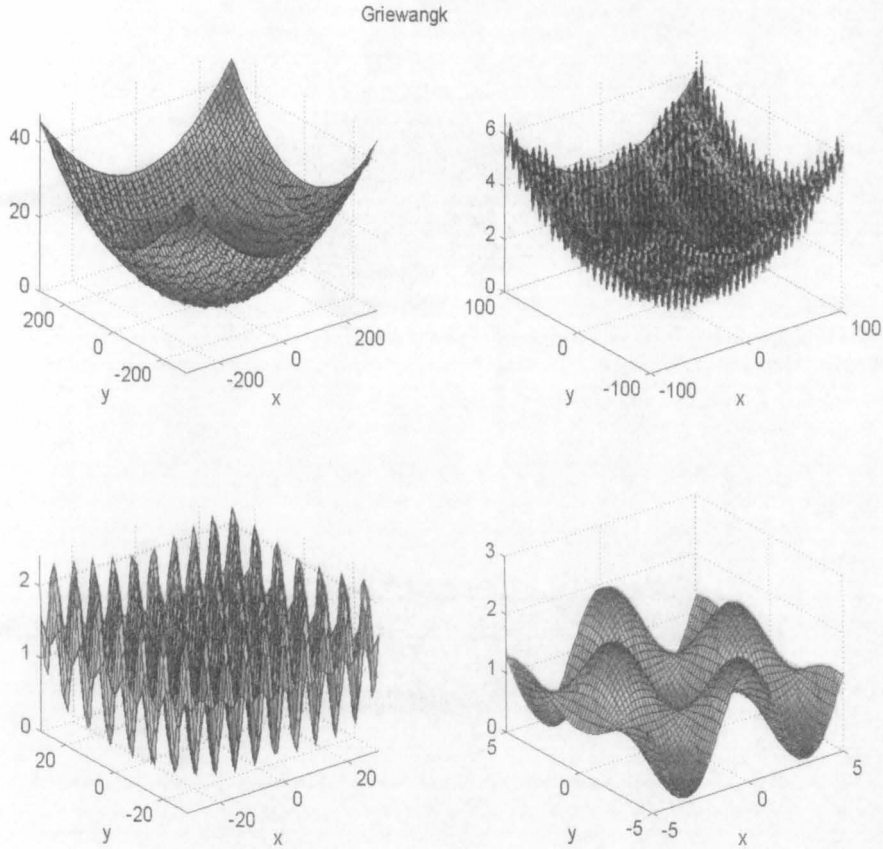


Figura 3.11: La función de Griewangk en dos dimensiones.

Donde c_i, a_{ij} $(i = 1, ..., m, j = 1, ..., N)$ son constantes. Con frecuencia se toma $m = 5$. En la figura se tomó:

$$f(x,y) = \sum_{i=1}^{5} c_i \exp\left[-\frac{1}{\pi}(x-a_i)^2 - \frac{1}{\pi}(y-b_i)^2\right] \cos\left[\pi(x-a_i)^2 + \pi(y-b_i)^2\right]$$

donde $a = [3, 5, 2, 1, 7], b = [5, 2, 1, 4, 9], c = [1, 2, 5, 2, 3]$.

La figura 3.12, muestra la misma función pero en diferentes escalas.

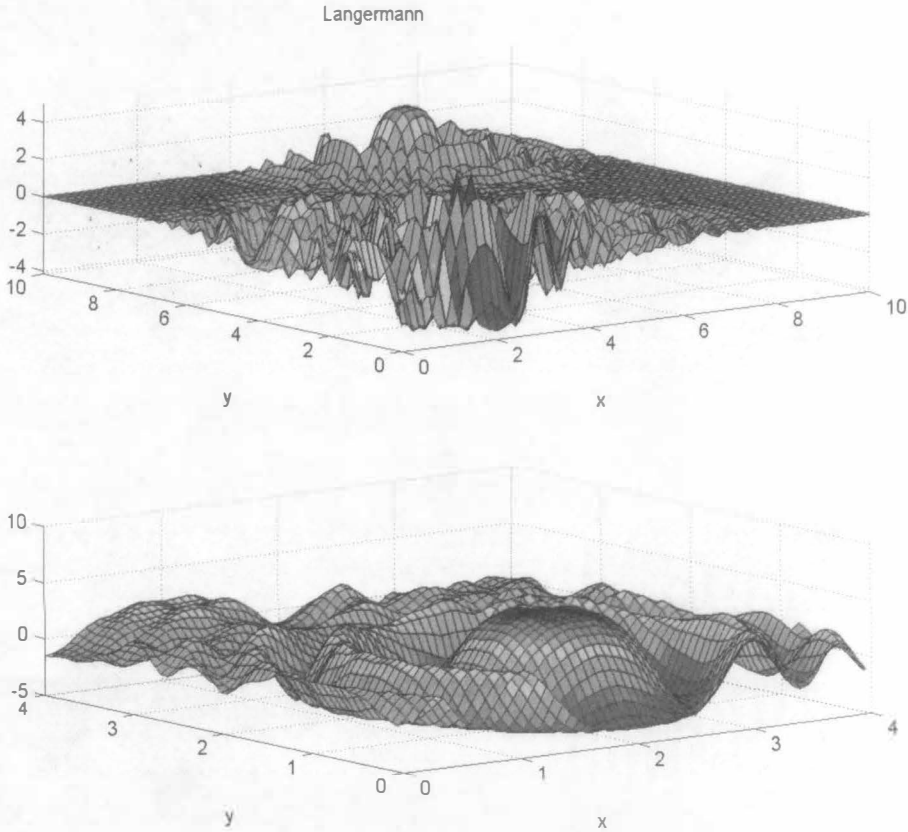


Figura 3.12: La función de Langermann en dos dimensiones con m = 5.

3.2.8 La función de Michalewicz

La función de Michalewicz es multimodal con un parámetro m, que determina la pendiente de los valles. Se define por:

$$f(x_1, x_2, \ldots, x_N) = -\sum_{i=1}^{N} \sin(x_i) \left[\sin(\frac{ix_i^2}{\pi})\right]^{2m}$$

$$0 \leq x_i \leq \pi, i = 1, 2, \ldots, N.$$

Se ha aproximado el mínimo global por $f(x_1, x_2, \ldots, x_5)$ = -4.687, y $f(x_1, x_2, \ldots, x_{10})$ = -9.66.

La figura 3.13, muestra la función con diferentes valores de m.

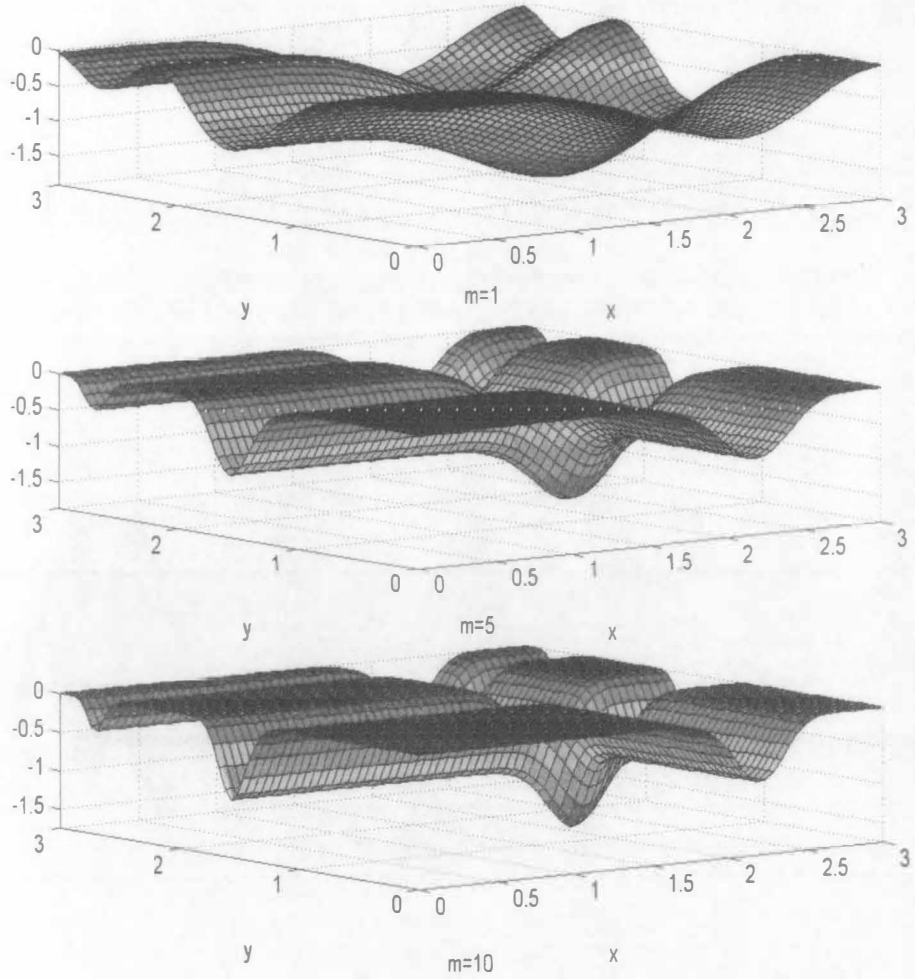


Figura 3.13: La función de Michalewicz en dos dimensiones para $m = 1$, 5, 10.

3.2.9 La función de Rastrigin

La función de Rastrigin es multimodal con muchos mínimos locales que están distribuidos de manera regular. Es una función similar a la función de Griewangk. Está definida por:

$$f(x_1, x_2, \ldots, x_N) = 10N + \sum_{i=1}^{N} [x_i^2 - 10\cos(2\pi x_i)],$$

$$-5.12 \leq x_i \leq 5.12, i = 1, 2, \ldots, N.$$

Tiene un mínimo global de $f(x_1, x_2, \ldots, x_N) = 0$ en $x_i = 0, i = 1, ..., N$.

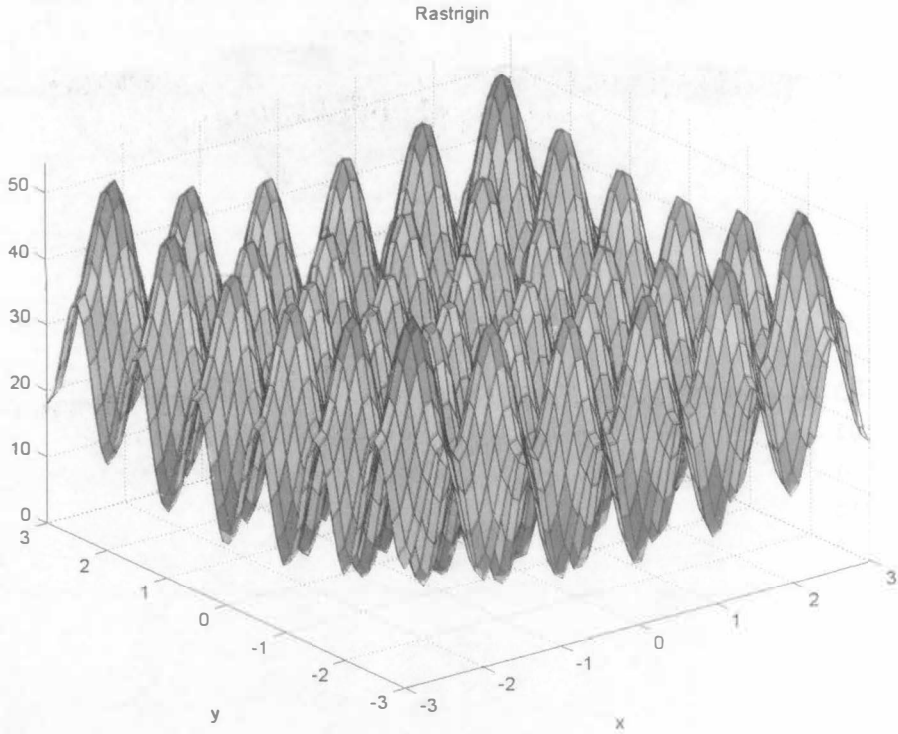


Figura 3.14: La función de Rastrigin en dos dimensiones.

3.2.10 La función de Rosenbrock

Conocida como el valle de Rosenbrock, o la función plátano. El mínimo global está dentro de una valle largo y angosto, y es difícil de encontrar. La definición está dada por:

$$f(x_1, x_2, \ldots, x_N) = \sum_{i=1}^{N-1} \left[(1 - x_i)^2 + 100(x_{i+1} - x_i^2)^2\right]$$

$$-2.048 \leq x_i \leq 2.048.$$

Tiene mínimo global de $f(x_1, x_2, \ldots, x_N) = 0$ en $x_i = 1$, $i = 1, ..., N$.

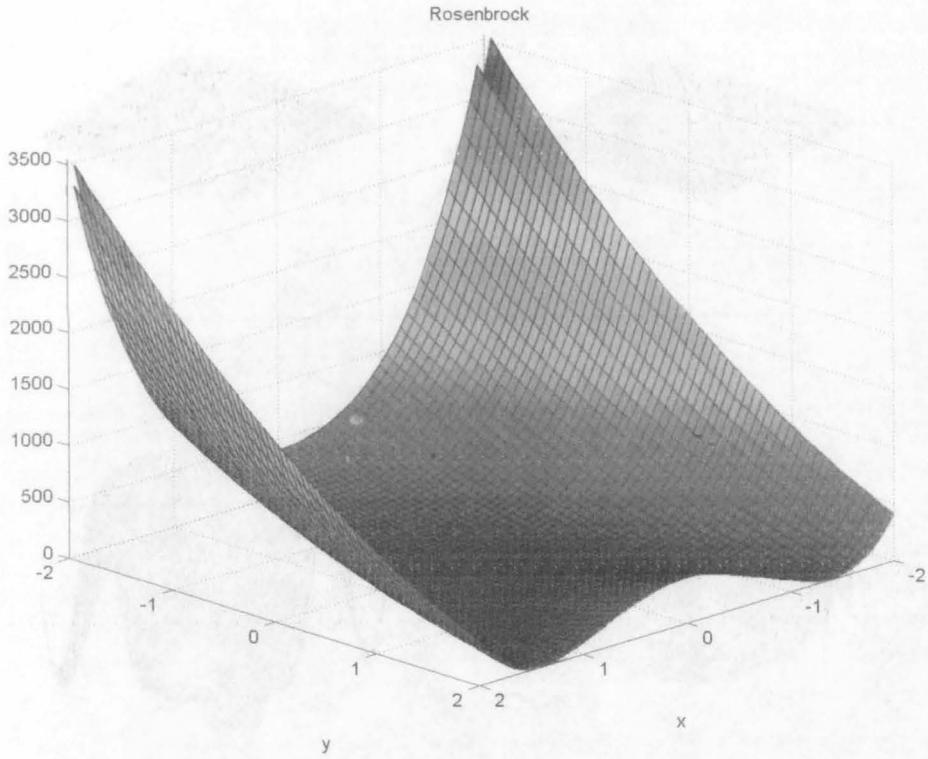


Figura 3.15: El valle de Rosenbrock en dos dimensiones.

3.2.11 La función de Schaffer F6

La función F6 de Schaffer es una función multimodal definida en dos dimensiones con un solo mínimo global. Está definida por:

$$f(x, y) = 0.5 + \frac{(\sin(\sqrt{x^2 + y^2}))^2 - 0.5}{(1 + 0.001(x^2 + y^2))^2}$$

$$-100 \leq x, y \leq 100.$$

Tiene un mínimo global de $f(x, y) = 0$ en $x = y = 0$.

La figura 3.16, muestra la misma función pero en diferentes escalas.

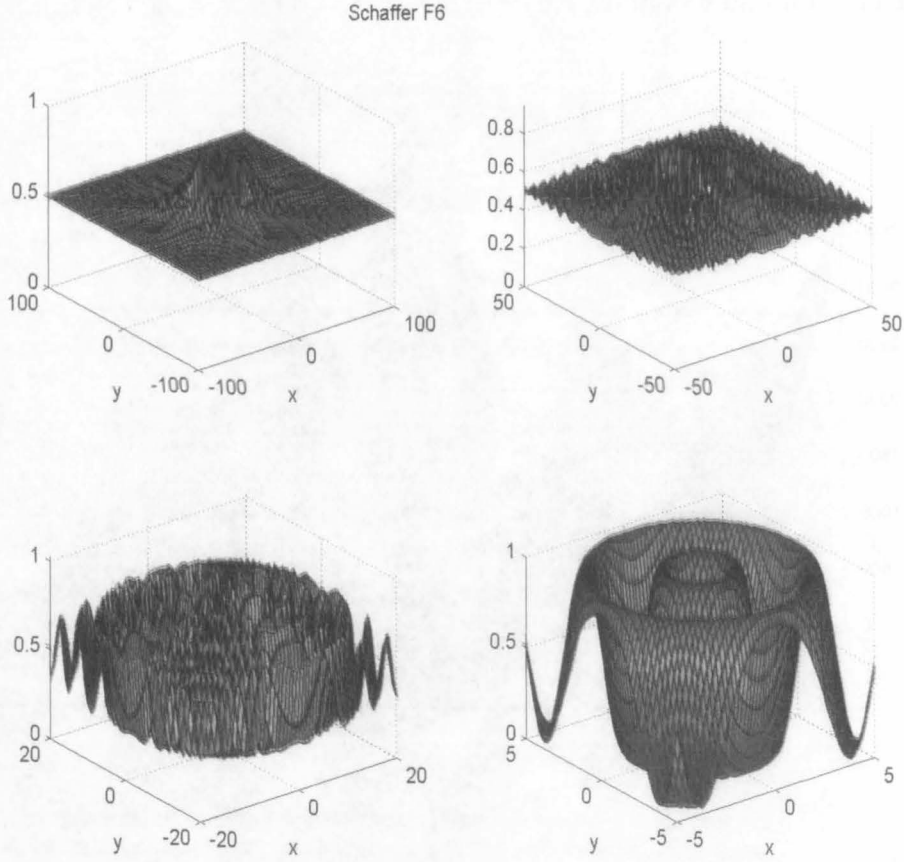


Figura 3.16: La función de Schaffer.

3.2.12 La función de Schwefel

La función de Schwefel es multimodal con un mínimo global. Es una función difícil para algoritmos de búsqueda, porque el siguiente mínimo local más pequeño no esta cerca al global. Está definida por:

$$f(x_1, x_2, \ldots, x_N = 418.9829N - \sum_{i=1}^{N}[x_i \sin(\sqrt{|x_i|})]$$

$$-500 \leq x_i \leq 500, i = 1, 2, \ldots, N.$$

Tiene un mínimo global de $f(x_1, x_2, \ldots, x_N) = 0$ en $x_i = 420.9687$, $i = 1, ..., N$.

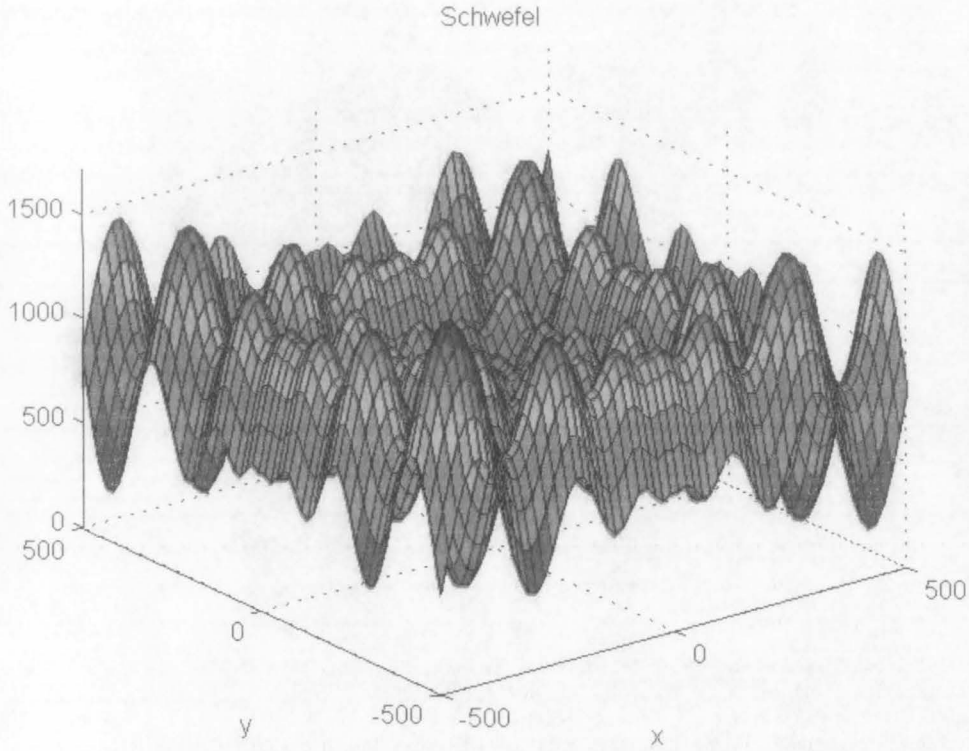


Figura 3.17: La función de Schwefel.

3.2.13 Las seis jorobas del camello

La función conocida como las seis jorobas del camello tiene un mínimo global en dos puntos diferentes, y cuatro otros mínimos locales. Está definida por:

$$f(x,y) = (4 - 2.1x^2 + x^{4/3})x^2 + xy + (-4 + 4y^2)y^2,$$
$$-3 \leq x \leq 3, \quad -2 \leq y \leq 2.$$

Tiene un mínimo global de $f(x,y)$ = -1.0316 en los dos puntos: x = -0.0898, y = 0.7126 y x = 0.0898, y = -0.7126.

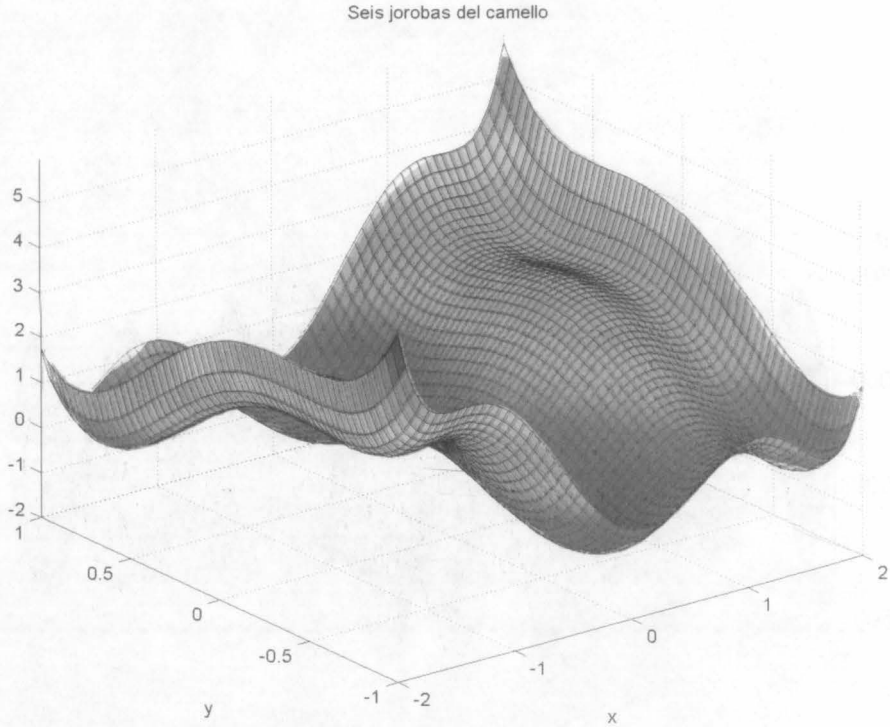


Figura 3.18: La función de las seis jorobas del camello.

Capítulo 4

El Algoritmo de Recocido Simulado

El propósito de este capítulo es presentar un algoritmo general para la solución de problemas de optimización combinatoria denominado *recocido simulado* (en inglés *simulated annealing*). Este algoritmo tiene la propiedad de que, en el límite, converge al conjunto de soluciones óptimas del problema de optimización combinatoria y en un tiempo polinomial produce buenas soluciones.

La técnica de recocido simulado fue propuesta por Kirkpatrick, Gellat y Vecchi en 1983 [67] y originalmente se basó en una analogía entre la simulación de recocido de sólidos y el reto de resolver problemas de optimización combinatoria de gran escala. Por esta razón el algoritmo se conoce como recocido simulado. Aquí se da esta analogía con el fin de introducir el algoritmo de recocido simulado desde un punto de vista intuitivo.

4.1 Recocido Simulado

La característica sobresaliente de la técnica es su aplicación general y la habilidad para obtener soluciones arbitrariamente cercanas a la óptima; sin embargo, el obtener soluciones de alta calidad puede requerir de mucho esfuerzo computacional. Una comparación importante que puede hacerse con respecto a si una técnica es general o no lo es, es la que se presenta en la programación lineal y la programación dinámica. La programación lineal es un modelo muy específico para solucionar problemas en donde la función ob-

jetivo y las restricciones deben ser lineales; si el problema no cae dentro de este esquema restringido, la programación lineal no puede aplicarse. Por otro lado, la programación dinámica se puede usar para resolver muchos problemas de optimización y su aplicabilidad depende de la habilidad que se tenga para definir las etapas del problema, las ecuaciones recursivas, las variables de estado y las variables de decisión para un problema específico. En este sentido, recocido simulado resulta ser análogo a la programación dinámica. El uso y la calidad de sus resultados dependen del arte y la habilidad con que se definan sus diferentes componentes como son: las evaluaciones de la función objetivo de la nueva solución con respecto a la anterior, la estructura de vecindades y el programa de enfriamiento; así como también, el grado de refinamiento de la implantación en la computadora.

4.1.1 El Proceso de Recocido de un Sólido

Recocido denota un proceso de calentamiento de un sólido a una temperatura en la que sus granos deformados recristalizan para producir nuevos granos. La temperatura de recocido o de recristalización, depende del tipo de material, del grado de deformación del mismo, además de su uso futuro. Seguida a la fase de calentamiento, viene un proceso de enfriamiento en donde la temperatura se baja poco a poco. De esta manera, cada vez que se baja la temperatura, las partículas se reacomodan en estados de más baja energía hasta que se obtiene un sólido con sus partículas acomodadas conforme a una estructura de cristal. Si se comienza con un valor máximo de la temperatura, en la fase de enfriamiento del proceso de recocido, para cada valor de la temperatura T debe permitirse que se alcance su equilibrio térmico; si el proceso de enfriamiento es demasiado rápido y no se alcanza en cada etapa el equilibrio térmico, el sólido congelará en un estado cuya estructura será amorfa en lugar de la estructura cristalina de más baja energía, esta estructura está caracterizada por una imperfecta cristalización del sólido.

El equilibrio térmico está caracterizado por la distribución de Boltzmann [101]. De acuerdo a esta distribución, la probabilidad de que el sólido esté en un estado i con energía E_i a la temperatura T, viene dada por

$$P_T\{X = i\} = \frac{1}{Z(T)} \exp\left(\frac{-E_i}{k_B T}\right) \tag{4.1}$$

donde X es una variable aleatoria que denota el estado actual del sólido. $Z(T)$ es una constante de normalización llamada la *función partición*, que

está definida como

$$Z(T) = \sum_j \exp\left(\frac{-E_j}{k_B T}\right) \tag{4.2}$$

donde la sumatoria se extiende sobre todos los posibles estados y k_B es una constante física conocida como la constante de Boltzmann. El factor $\exp\left(\frac{-E_i}{k_B T}\right)$ se conoce como el factor de Boltzmann. Obviamente $P_T\{X = i\}$ es una función de densidad de probabilidad ya que siempre es mayor o igual a cero y la suma sobre todos los valores es igual a la unidad. Se puede observar en $P_T\{X = i\}$ que cuando el valor de T disminuye, la distribución de Boltzmann se concentra en los estados de menor energía, mientras que si la temperatura se aproxima a cero, únicamente los estados con mínima energía tienen una probabilidad de ocurrencia diferente de cero.

Por lo dicho anteriormente, el proceso de recocido consta de dos pasos fundamentales que son:

1. Incrementar la temperatura del baño térmico a un valor máximo.

2. Decrementar *cuidadosamente* la temperatura del baño térmico hasta que las partículas se reacomoden por sí mismas en un estado de mínima energía, denominado el *estado fundamental del sólido.*

En el proceso de elevar la temperatura del sólido, todas las partículas se reacomodan aleatoriamente. En el estado fundamental, las partículas se acomodan en una retícula altamente estructurada y la energía del sistema es mínima. El estado fundamental del sólido se obtiene solamente si la temperatura máxima es suficientemente elevada y el proceso de enfriamiento es suficientemente bajo. De otra manera se obtendría un estado amorfo denominado *meta-estado.*

El proceso físico de recocido puede modelarse exitosamente usando métodos de simulación (vea [79]), el algoritmo introducido para tal propósito se basa en técnicas Monte Carlo y genera una sucesión de estados del sólido de la siguiente manera: Dado un estado i del sólido con energía E_i, se genera el estado siguiente j aplicando un mecanismo de perturbación que transforma el estado actual en el siguiente estado por medio de una pequeña distorsión, por ejemplo, por el desplazamiento de una partícula. La energía del siguiente estado es E_j. Si la *diferencia de energía*, $E_j - E_i$, es menor o igual a cero, el

estado j se acepta como el estado actual. Si la diferencia de energía es mayor que cero, el estado j se acepta con una probabilidad que está dada por:

$$\exp\left(\frac{E_i - E_j}{k_B T}\right), \tag{4.3}$$

donde T denote la temperatura del baño térmico y k_B es la constante de Boltzmann. La regla de decisión descrita arriba se conoce como el *criterio de Metropolis* y al algoritmo se le conoce como *algoritmo de Metropolis*.

Si la temperatura se baja poco a poco, el sólido puede alcanzar su equilibrio térmico en cada temperatura. En el algoritmo de Metropolis esto se lleva a cabo generando un número grande de transiciones para un valor dado de la temperatura.

4.1.2 El Algoritmo de Recocido Simulado

La simulación del proceso de recocido puede usarse para describir un proceso de generación de sucesiones de soluciones de un problema de optimización combinatoria en donde se vaya obteniendo, conforme el proceso avanza, mejores soluciones al mismo. Para este propósito, se puede observar una analogía entre el sistema físico y un problema de optimización combinatoria en donde cada solución del problema puede verse como un estado del sólido y los valores de la función objetivo para cada solución como los niveles de energía del sólido. En resumen, se puede pensar en las siguientes equivalencias:

1. Las soluciones de un problema de optimización combinatoria son equivalentes a los estados de un sistema físico.

2. El costo de una solución es equivalente a la energía de un estado.

Además, se introduce un parámetro que juega el papel equivalente de la temperatura. Este parámetro se llama el *parámetro de control*. El algoritmo de recocido simulado puede verse como una iteración del algoritmo de Metropolis, evaluado en valores decrecientes del parámetro de control.

A continuación se introducen las siguientes definiciones.

Definición 4.1. *Sea (S, F) denote una instancia de un problema de optimización combinatoria y denote por i y j dos soluciones con costo f(i) y f(j), respectivamente. Entonces, el criterio de aceptación determina si j se acepta a partir de i al aplicar la siguiente probabilidad de aceptación:*

$$P_c\{\text{aceptar } j\} = \begin{cases} 1 & si\ f(j) \leq f(i) \\ \exp\left(\frac{f(i)-f(j)}{c}\right) & si\ f(j) > f(i), \end{cases} \tag{4.4}$$

donde $c \in R^+$ denota el parámetro de control.

Claramente, el mecanismo de generación corresponde al mecanismo de perturbación en el algoritmo de Metropolis, mientras que el criterio de aceptación corresponde al denominado criterio de Metropolis dado por la ecuación 4.4.

Definición 4.2. *Una transición es una acción combinada que transforma la solución actual en una subsecuente. Esta acción consiste de los siguientes dos pasos: (i) aplicación del mecanismo de generación, (ii) aplicación del criterio de aceptación.*

Sea c_k denote el valor del parámetro de control y L_k el número de transiciones generadas en la k-ésima iteración del algoritmo de Metropolis. De este modo, el algoritmo de recocido simulado puede describirse en pseudocódigo como se muestra en la Figura 4.1.

El algoritmo de Recocido Simulado comienza llamando a un procedimiento de inicialización donde se definen la solución inicial, parámetro de control inicial y el número inicial de generaciones necesarias para alcanzar el equilibrio térmico para la temperatura inicial. La parte medular del algoritmo consta de dos ciclos. El externo **Repite**...**hasta** y el interno **Para**...**finpara**. El ciclo interno mantiene fijo el parámetro de control hasta que se generan L_k soluciones y se acepta o se rechaza la solución generada conforme los criterios de aceptación ya discutidos. El ciclo externo disminuye el valor de la temperatura mediante el procedimiento CALCULA-CONTROL y calcula el número de soluciones a generar para alcanzar equilibrio térmico mediante el procedimiento CALCULA-LONGITUD. Este ciclo finaliza cuando la condición de paro se cumple.

Comienza

INICIALIZA $((i_{inicial}, c_0, L_0)$

$k := 0$

$i: = i_{inicial}$

Repite

Para $l :=$ a L_k **hacer**

Comienza

GENERA (j de S_i)

si $f(j) \leq f(i)$ **entonces** $i := j$

sino

si $\exp\left(\frac{f(i)-f(j)}{c_k}\right)$ >número aleatorio en [0,1) **entonces**

$i := j$

finpara

$k := k + 1$

CALCULA-LONGITUD (L_k)

CALCULA-CONTROL (c_k)

hasta criterio de paro

fin

Figura 4.1: Algoritmo de Recocido Simulado en pseudocódigo

Un rasgo característico del algoritmo de recocido simulado es que, además de aceptar soluciones donde se disminuye el costo, también acepta donde el costo aumenta. Inicialmente, para valores grandes de c, puede aceptar soluciones deterioradas donde el costo está muy por encima del valor de la solución actual; cuando c decrece, únicamente pequeñas desviaciones serán aceptadas y finalmente, cuando el valor de c se aproxima a cero, no se aceptarán desviaciones. Este hecho significa que el algoritmo de recocido simulado tiene la capacidad de escapar de mínimos locales.

Note que la probabilidad de aceptar desviaciones está implementada al comparar el valor de $\exp((f(i) - f(j))/c$ con un número aleatorio generado de una distribución uniforme en el intervalo [0,1). Además, debe ser obvio que la velocidad de convergencia del algoritmo está determinada al escoger los parámetros L_k y c_k, $k = 0,1,\ldots$. Si los valores c_k decrecen de manera acelerada o los valores de L_k no son tan grandes, se tendrá una convergencia más rápida que cuando los valores de c_k decrecen lentamente o los valores de L_k son grandes.

Comparando el algoritmo de recocido simulado con el algoritmo de búsqueda local, resulta evidente que recocido simulado puede verse como una generalización de búsqueda local y viene a ser idéntico a búsqueda local, en el caso de que el parámetro de control c se tome igual a cero. En lo que respecta a la ejecución de ambos algoritmos, generalmente, el algoritmo de recocido simulado obtiene soluciones de mejor calidad que el algoritmo de búsqueda local.

4.1.3 Aspectos Generales del Algoritmo

En las aplicaciones del algoritmo de recocido simulado, comúnmente se desea implementar éste, de manera que la sucesión de soluciones estén generadas a partir de valores decrecientes del parámetro de control. Las soluciones se generan continuamente tratando de transformar la solución actual en una subsecuente por medio de la aplicación del mecanismo de generación y el criterio de aceptación. Las aplicaciones del algoritmo de recocido simulado requieren de la especificación de los siguientes puntos: (a) Una representación concisa del problema, (b) un mecanismo de transición y (c) un programa de enfriamiento. Cada uno de estos puntos se pueden enumerar con más detalle.

1. Una descripción concisa de la *representación del problema* consiste de una representación del espacio de soluciones y una expresión de la función de costo. La función de costo debe escogerse de manera que represente la efectividad de las soluciones con respecto al objetivo de optimización. La representación del problema y la función de costo deben darse por medio de expresiones simples que permitan una fácil manipulación.

2. La generación de ensayos para transformar la solución actual en una subsecuente consiste de tres pasos. Primero, se debe generar una nueva solución aplicando un mecanismo de generación; enseguida, se debe calcular la diferencia de costo de las dos soluciones; por último, se hace una decisión de aceptar o no, la nueva solución. La evaluación de la nueva solución es lo que más tiempo consume en el algoritmo de recocido simulado y por lo tanto debe hacerse lo más eficientemente posible. El mecanismo de generación usualmente se escoge de tal manera que se obtengan modificaciones simples para que puedan ser ejecutadas rápidamente, por ejemplo permutaciones, cambios o inversiones. El cálculo de la diferencia de costo se hace tomando en cuenta las diferencias entre ambas soluciones. Para muchas aplicaciones éste es el camino más rápido para calcular las diferencias en el costo. La decisión de aceptar la nueva solución se basa en un criterio de aceptación (vea Definición 4.1).

3. Ejecutar el proceso de recocido, requiere de la especificación de los parámetros que determinan el programa de enfriamiento. Se debe dar un valor inicial del parámetro de control, una función que especifique su decremento, la longitud de cada bloque donde permanece constante el parámetro de control y el criterio de paro.

En [2] se muestra que si se selecciona el mecanismo de generación, el criterio de aceptación como en la Definición 4.1, una estructura de vecindades y un programa de enfriamiento adecuado; el algoritmo de recocido simulado converge con probabilidad 1 al conjunto de soluciones óptimas.

4.2 Ejemplo: El Problema de Conjunto Independiente

El objetivo fundamental de esta sección es presentar la bondad de la técnica de recocido simulado para el problema de conjunto independiente dado en la Sección 3.1.4, así como los lineamientos a seguir para implantarlo en la computadora.

4.2.1 Algoritmo de Recocido Simulado

El algoritmo de recocido simulado puede usarse para atacar este problema y obtener soluciones que no necesariamente son óptimas, pero se puede mostrar un comportamiento muy favorable del método con respecto a instancias en las que se conoce la solución. Para aplicar el algoritmo de recocido simulado se necesitan definir ciertos puntos que más adelante se describen.

Para el caso del problema de conjunto independiente, se tiene que agregar adicionalmente a la función objetivo, un costo para cuando se tengan soluciones infactibles; esto es, subconjuntos de V que no necesariamente son conjuntos independientes. Para este problema es apropiado partir la familia de conjuntos de todos los subconjuntos de V en dos subfamilias; la familia de soluciones factibles, es decir, la familia de conjuntos que son independientes y la familia de subconjuntos que no son independientes. La infactibilidad de algunas soluciones puede usarse ventajosamente por el algoritmo de recocido simulado para evitar quedar atrapado en un conjunto independiente maximal de menor cardinalidad a la del máximo.

4.2.2 Descripción del Algoritmo

El algoritmo de recocido simulado puede aplicarse de la siguiente manera:

1. El espacio de soluciones S consiste en todas las posibles particiones del conjunto V en subconjuntos V' y $V - V'$. El conjunto V' consiste en todas las particiones donde $\forall u, v \in V' : (u, v) \notin E$.

2. La función de costo, que debe ser maximizada, se escoge como

$$f\left(V'\right) = \left|V'\right| - \lambda\left|E'\right|, \tag{4.5}$$

donde E' denota el conjunto de arcos $(u, v) \in E$ con $u, v \in V'$ y λ denota el *factor de peso* que debe ser mayor que 1 y juega el papel de penalización en el algoritmo. Las soluciones factibles, únicamente contribuyen en el primer término de la función de costo. Soluciones infactibles contribuyen en el segundo término y por lo tanto bajan el costo de la partición para puntos con igual cardinalidad. Así, el segundo término puede verse como una función de penalización que castiga la presencia de arcos entre nodos de la partición.

La generación de nuevas soluciones se hace de manera aleatoria, escogiendo un nodo $u' \in V$ y cambiándolo de V' a $V - V'$ si $u' \in V'$ o viceversa. La diferencia en costo viene dada por la función:

$$\Delta f = \left(\chi_{(V-V')}(u') - \chi_{(V')}(u')\right)\left(1 - \lambda \sum_{(u',v)\in E, v\in V'} 1\right). \tag{4.6}$$

donde la función $\chi_{(V)}(u) = 1$ si $u \in V$ y vale cero en otro caso.

Claramente, el espacio de soluciones para el conjunto independiente puede hacerse restrictivo solamente al conjunto de soluciones factibles. Además, la presencia de soluciones infactibles permite que se defina la función de costo de manera suave haciendo que el algoritmo de recocido simulado escape fácilmente de óptimos locales ya que existen más soluciones con diferencias pequeñas en costo en la vecindad de un óptimo local y así se incrementa la probabilidad de escape. Agrandar el conjunto de soluciones factibles con el conjunto de soluciones infactibles en este caso (como en muchos) hace posible el uso de un mecanismo de generación simple que a menudo permite una convergencia rápida. El algoritmo que describe el procedimiento de recocido simulado puede enunciarse de la manera siguiente, dado en la Figura 4.2.

4.2.3 Experiencia Computacional

La prueba del algoritmo se llevó a cabo generando de manera aleatoria gráficas conectadas. Para valores preestablecidos del número n de nodos y el número m de arcos, se generaron 100 gráficas aleatorias. La Tabla

Propósito: Obtiene un conjunto independiente maximal de la mayor cardinalidad posible de la gráfica $G = (V, E)$ dada.

Descripción:

1. Tome c un valor grande apropiado.
2. Tome $V' \subset \{1, 2, \ldots, n\}$ arbitrario.
   Obtenga el valor correspondiente de la función objetivo $f(V')$.
3. Asigne el valor verdadero a la variable lógica CAMBIO.
4. Si la variable CAMBIO tiene valor falso, entonces pare.
5. Haga CAMBIO igual a falso.
6. Repita los puntos (a)-(i) L_c veces.
   (a) Seleccione un número aleatorio u' con distribución uniforme en $\{1, 2, \ldots, n\}$.
   (b) Evalué el valor Δf de la expresión 4.6.
   (c) Si $\Delta f < 0$ entonces vaya a (g).
   (d) Sea $P(\Delta f) = \exp(-\Delta f / c)$.
   (e) Genere un número aleatorio x, uniformemente distribuido en (0,1).
   (f) Si $x \geq P(\Delta f)$ entonces vaya a (i).
   (g) Haga $V' \leftarrow V' \bigcup \{u'\}$ si $u' \notin V'$ o $V' \leftarrow V' - \{u'\}$ si $u' \in V'$.
   (h) $\Delta f \neq 0$ entonces CAMBIO $\leftarrow$ verdadero.
   (i) Fin del paso 6.
7. Reduzca c e incremente r.
8. Vaya al paso 4.

Figura 4.2: Algoritmo de Recocido Simulado para el Problema de Conjunto Independiente Máximo.

Num. Vértices	RECOCIDO SIMULADO Num. Aristas / Tiempo(seg.)			
	15	20	30	40
10	2.05	2.64	3.20	4.48
	30	50	70	90
15	2.16	3.15	4.47	5.50
	40	80	120	160
20	3.22	4.83	5.69	6.96
	60	120	240	350
30	3.57	5.78	6.17	7.26
	80	240	320	600
40	4.24	5.89	6.69	7.94
	100	200	400	800
50	6.13	6.78	7.17	8.31

Tabla 4.1: Tiempos promedio de ejecución de recocido simulado para el problema de conjunto independiente.

4.1 reporta el tiempo promedio (en segundos) y los valores del número de vértices y de aristas considerados. Para cada valor del número de vértices, se consideraron cuatro casos (en cuanto a número de aristas): gráfica con pocas aristas, gráfica con un número normal de aristas, gráfica medianamente densa y gráfica densa.

Cabe mencionar que el tiempo de solución del algoritmo de recocido simulado es muy variable, ya que depende en gran medida del valor inicial del parámetro de control c, del número r de veces que se ejecuta el paso 6 (del algoritmo) y de la forma en que se reduzca c y se incremente r. Para el análisis realizado aquí, se tomó un valor inicial de c de tal manera que la tasa de aceptación de las permutaciones propuestas fuera mayor o igual a 0.8 y los decrementos fueran del 0.1 (paso 7 del algoritmo). El número r inicial que se tomó en cada caso fue $r = n$ y los incrementos fueron de 0.05 (paso 7 del algoritmo).

4.2.4 Análisis Comparativo con un Método Exacto

Con el fin de hacer una comparación del algoritmo de recocido simulado con respecto a un algoritmo exacto, se tomó un algoritmo enumerativo reportado por Christofides [25] que, según estudios de Syslo [100], es uno de

Num. Vértices	EXACTO			
	Num. Aristas / Tiempo(seg.)			
10	15 0.01	20 0.01	30 0.01	40 0.01
15	30 0.01	50 0.04	70 0.05	90 0.03
20	40 0.05	80 0.14	120 0.33	160 0.29
30	60 0.53	120 3.02	240 21.13	350 8.20
40	80 2.57	240 336.82	320 125.00	600 4648.49
50	100 40.18	200 1354.57	400 7326.73	800 —

Tabla 4.2: Tiempos promedio de ejecución del algoritmo exacto para el problema de conjunto independiente.

los métodos exactos más "eficientes" con que se cuenta en la actualidad para resolver el problema. Los tiempos obtenidos por este algoritmo, tomando exactamente las mismas gráficas que se usaron con el algoritmo de recocido simulado se reportan en la Tabla 4.2.

En la Tabla 4.2 se puede observar el crecimiento dramático del tiempo en cuanto aumentan el número de vértices y aristas en la gráfica. Para 50 vértices y 800 aristas el algoritmo no obtuvo la solución después de 24 horas. También se calculó la *eficiencia del algoritmo* de recocido simulado en cuanto a la *calidad de solución* o cercanía al óptimo, por medio de la ecuación

$$eficiencia = 1 - \frac{z - z_{opt}}{z_{opt}} \tag{4.7}$$

donde z denota el valor de la función objetivo en la solución obtenida y z_{opt} es el valor óptimo de la instancia del problema. Los cálculos que aparecen en la Tabla 4.3 son con base en los promedios obtenidos en las 100 instancias resueltas para cada valor del número de arcos m y del número de nodos n de los valores en la función objetivo de la solución correspondiente. En la Tabla 4.3, puede observarse que las eficiencias del algoritmo de recocido simulado

		ALGORITMO (eficiencia)	
n	m	EXACTO	REC. SIM.
10	15	1.00	0.9954
	20	1.00	0.9944
	30	1.00	0.9256
	40	1.00	0.9903
15	30	1.00	0.9830
	50	1.00	0.9886
	70	1.00	0.9871
	90	1.00	0.9827
20	40	1.00	0.9865
	80	1.00	0.9813
	120	1.00	0.9818
	160	1.00	0.9771
30	60	1.00	0.9892
	120	1.00	0.9867
	240	1.00	0.9810
	350	1.00	0.9754
40	80	1.00	0.9834
	240	1.00	0.9823
	320	1.00	0.9942
	600	1.00	0.9725
50	100	1.00	0.9791
	200	1.00	0.9821
	400	1.00	0.9802

Tabla 4.3: Comparación de los algoritmos que resuelven el problema de conjunto independiente con respecto a su eficiencia.

son cercanas a 1, una ventaja que tiene, es que cada vez que se ejecutado da una solución diferente y por lo tanto cuando se ejecuta varias veces se tiene una probabilidad de 1.0 de obtener la solución óptima del problema.

Las pruebas computacionales con el problema de conjunto independiente, revelan un buen comportamiento del algoritmo. Las soluciones subóptimas difieren en 1-2% de la solución óptima. Además si se ejecuta el procedimiento varias veces con diferentes valores iniciales, se obtiene en todos los casos la solución óptima. Además de que la implantación del algoritmo en la computadora, no requiere de gran capacidad de memoria y permite un tiempo razonable para obtener buenas soluciones en las computadoras personales.

4.3 Ejemplo: Problema de Asignación Cuadrático

El Problema de Asignación Cuadrática, vea la Sección 3.1.3, es una variante importante del problema de localización clásico cuando el costo asociado por colocar una planta en un determinado lugar depende no únicamente de las distancias entre plantas y de las demandas respectivas, sino también de la interacción con otras plantas.

4.3.1 Algoritmo de Recocido Simulado

El algoritmo de recocido simulado puede usarse para atacar este problema y obtener soluciones que no necesariamente son óptimas pero se puede mostrar un comportamiento muy favorable del método con respecto a instancias que se conoce la solución, así como para aquellas instancias que no se conoce la solución óptima pero se conoce una mejor solución.

Para aplicar el algoritmo de recocido simulado se necesitan definir ciertos puntos que a continuación se describen.

Como ya se ha mencionado, el espacio de soluciones consiste de todas las permutaciones de n elementos. Para cada permutación π, se define la función de costo como:

$$f = \sum_{i=1}^{n} \sum_{i=1}^{n} a_{ip} b_{\pi(i)\pi(p)} \tag{4.8}$$

La generación de nuevas soluciones se hace a partir de tomar permutaciones que difieran de la actual en únicamente dos cambios. Se comienza con una permutación π y se seleccionan dos índices i y k y a partir de estos índices se define la nueva permutación π por:

$$\begin{aligned} \overline{\pi}(i) &\equiv \pi(k), \quad \overline{\pi}(k) \equiv \pi(i) \\ \overline{\pi}(l) &\equiv \pi(l), \quad l = 1, 2, \ldots, n, \ l \neq i, k. \end{aligned} \tag{4.9}$$

La diferencia en la función de costo entre la solución actual y la nueva solución generada a partir del cambio i k puede calcularse de la siguiente expresión:

$$\begin{aligned} \Delta f = \sum_{l=1, l\neq i,k} & [(a_{il} - a_{kl})(b_{\pi(k)\pi(l)} - b_{\pi(i)\pi(l)}) + (a_{li} - a_{lk})(b_{\pi(l)\pi(k)} - b_{\pi(l)\pi(i)})] \\ & +(a_{ii} - a_{kk})(b_{\pi(k)\pi(k)} - b_{\pi(i)\pi(i)}) \\ & +(a_{ik} - a_{ki})(b_{\pi(k)\pi(i)} - b_{\pi(i)\pi(k)}) \end{aligned} \tag{4.10}$$

donde Δf denota la diferencia en el costo entre la solución propuesta menos la solución actual al hacer el intercambio. Si $\Delta f < 0$, entonces $\overline{\pi}$ tiene un valor en la función objetivo menor que π. Usando estas observaciones se puede establecer el algoritmo de recocido simulado para el problema de asignación cuadrática como aparece en la Figura 4.3.

Experiencia Computacional

La prueba del algoritmo de recocido simulado se llevó a cabo generando instancias del problema de asignación cuadrática de manera aleatoria. Para la matriz A se generaron números aleatorios en el intervalo $[1, 100]$ y para la matriz B se generaron números aleatorios en el intervalo $[1, 50]$. Para cada valor de $n = 5$, 6, 7, 8, 10, 12, 15, 20, 25 y 30 se generaron 100 instancias de manera aleatoria. La Tabla 4.4 reporta el tiempo promedio (en segundos).

Adicionalmente, para $n = 5$, 6, 7, 8, 12, 15, 20 y 30 se corrieron los problemas de prueba de Nugent (vea [89]). Los resultados de estas corridas aparecen en la Tabla 4.5.

Propósito: Obtener una permutación π que minimice el valor de la función de la ecuación 4.8.

Descripción:

1. Tome c un valor grande apropiado.
2. Obtenga una permutación aleatoria π del conjunto $\{1,2,\ldots,n\}$ Obtenga el valor correspondiente de la función objetivo $f(\pi)$.
3. Asigne el valor verdadero a la variable lógica CAMBIO.
4. Si la variable CAMBIO tiene valor falso, entonces pare.
5. Haga CAMBIO igual a falso.
6. Repita los puntos (a)-(i) r veces.
   (a) Tome una permutación arbitraria de dos cambios $\overline{\pi}$.
   (b) Evalué el valor Δf de la expresión 4.10
   (c) Si $\Delta f < 0$ entonces vaya a (g).
   (d) Sea $P(\Delta f) = \exp(-\Delta f/c)$.
   (e) Genere un número aleatorio x, con distribución uniforme (0,1).
   (f) Si $x \geq P(\Delta f)$ entonces vaya a (i).
   (g) Acepte la nueva permutación $\pi \leftarrow \overline{\pi}$, $f(\pi) \leftarrow f(\pi) + \Delta f$.
   (h) Si $\Delta f \neq 0$ entonces CAMBIO $\leftarrow$ verdadero.
   (i) Fin del paso 6.
7. Reduzca c e incremente r.
8. Vaya al paso 4.

Figura 4.3: Algoritmo de Recocido Simulado para el Problema de Asignación Cuadrática.

n	ALGORITMO REC. SIM. tiempo (seg.)
5	1.73
6	2.25
7	2.98
8	3.79
10	6.40
12	12.73
15	15.21
20	20.04
25	28.62
30	43.65

Tabla 4.4: Tiempos de ejecución del algoritmo de recocido simulado para el problema de asignación cuadrática.

ALGORITMO RECOCIDO SIMULADO			
n	Mejor Solución Conocida	Valor	Tiempo (seg.)
5	50 (óptima)	50	0.27
6	86 (óptima)	86	1.27
7	148 (óptima)	148	1.58
8	214 (óptima)	214	1.76
12	578 (óptima)	578	2.86
15	1150 (óptima)	1150	14.17
20	2570	2570	36.80
30	6124	6150	76.13

Tabla 4.5: Resultados obtenidos para los problemas de Nugent por el algoritmo de recocido simulado.

4.3.2 Algoritmo de Búsqueda Local

Uno de los puntos importantes del algoritmo de recocido simulado es el hecho de que cada que se ejecuta el paso 6 del mismo, no solamente acepta permutaciones π para las que se tenga un mejor valor objetivo, sino que también se aceptan con cierta probabilidad permutaciones cuyo valor objetivo es peor. Con el objeto de poder comparar los resultados del algoritmo de recocido simulado con el algoritmo de búsqueda local correspondiente, también se implantó este último en la computadora.

Siguiendo la misma notación que la ya dada para el algoritmo de recocido simulado, se puede describir el algoritmo de búsqueda local como aparece en la Figura 4.4.

Experiencia Computacional

La prueba del algoritmo se llevó a tomando las mismas instancias que para el algoritmo de recocido simulado. La Tabla 4.6 reporta el tiempo promedio (en segundos).
Adicionalmente, para $n = 5, 6, 7, 8, 12, 15, 20$ y 30 se corrieron los problemas de prueba de Nugent. Los resultados de estas corridas aparecen en la Tabla 4.7.

4.3.3 Análisis Comparativo de los Métodos

Como ya se mencionó, se generaron en forma aleatoria instancias del problema de asignación cuadrática. Para cada una de estas instancias generadas, se corrieron los cuatro algoritmos, un exacto, un heurístico hecho específicamente para el problema de asignación cuadrática y los anteriormente mencionados. Los resultados aparecen en la Tabla 4.8.

Cabe mencionar que el tiempo de solución del algoritmo de recocido simulado es muy variable, ya que depende en gran medida del valor inicial del parámetro de control c, del número r de veces que se ejecuta el paso 6, la forma en que se reduzca el parámetro c y se incremente el parámetro r. Para el análisis realizado en este documento, se tomó un valor inicial de c, de tal manera que la tasa de aceptación de las permutaciones propuestas fuera mayor o igual a 0.8 y los decrementos fueran del 0.1 (paso 7 del algoritmo). El número r inicial que se tomó en cada caso fue $r = 2 \times n$ y los incrementos

Propósito: Obtener una permutación π que minimice el valor de la función de la expresión 4.8.

Descripción:

1. Obtenga una permutación aleatoria π del conjunto $\{1,2,\ldots,n\}$. Obtenga el valor correspondiente de la función objetivo $f(\pi)$.
2. Asigne el valor verdadero a la variable lógica CAMBIO.
3. Si la variable variable CAMBIO tiene valor falso, entonces pare.
4. Haga CAMBIO igual a falso.
5. Repita los puntos (a)-(f) $n \times (n-1)$ veces.
   (a) Tome una permutación arbitraria de dos cambios $\overline{\pi}$.
   (b) Evalué el valor Δf de la expresión 4.10
   (c) Si $\Delta f \geq 0$ entonces vaya a (f).
   (d) Acepte la nueva permutación $\pi \leftarrow \overline{\pi}$, $f(\pi) \leftarrow f(\pi) + \Delta f$.
   (e) CAMBIO $\leftarrow$ verdadero.
   (f) Fin del paso 5.
6. Vaya al paso 3.

Figura 4.4: Algoritmo de Búsqueda Local para el Problema de Asignación Cuadrática.

n	ALGORITMO BUS. LOC. tiempo (seg.)
5	0.13
6	0.21
7	0.29
8	0.43
10	0.79
12	1.07
15	2.40
20	5.64
25	11.11
30	20.96

Tabla 4.6: Tiempos de ejecución del algoritmo de búsqueda local para el problema de asignación cuadrática.

ALGORITMO BÚSQUEDA LOCAL			
n	Mejor Solución Conocida	Valor	Tiempo (seg.)
5	50 (óptima)	50	0.11
6	86 (óptima)	86	0.22
7	148 (óptima)	148	0.28
8	214 (óptima)	214	0.33
12	578 (óptima)	592	1.32
15	1150 (óptima)	1178	2.41
20	2570	2644	5.27
30	6124	6354	15.81

Tabla 4.7: Resultados obtenidos para los problemas de Nugent por el algoritmo de búsqueda local.

fueron de 0.05 (paso 7 del algoritmo). También se calculó la eficiencia de cada uno de los algoritmos por medio de la ecuación 4.7. Esta ecuación se usó para los valores de $n = 5, 6, 7, 8, 10$ y 12. Para valores mayores de n, al no ser posible calcular la solución óptima, la eficiencia se calculó tomando como base el valor z_{rec}, esto es, el valor objetivo que da el algoritmo de recocido simulado, para el cual se usó la ecuación equivalente:

$$eficiencia = 1 - \frac{z - z_{rec}}{z_{rec}}$$

Los cálculos que aparecen en la Tabla 4.8 son con base en los promedios obtenidos en las 100 instancias resultas para cada valor de n de los valores en la función objetivo de la solución correspondiente.

En la Tabla 4.8, puede observarse que las eficiencias del algoritmo de recocido simulado son las más elevadas y los tiempos de ejecución del algoritmo de recocido simulado, crece de manera razonable; incluso, para instancias pequeñas, se obtienen tiempos de ejecución más grandes que para el algoritmo estocástico. Esta situación cambia para valores de n mayores.

Otra característica importante del algoritmo de recocido simulado es que las variaciones en las soluciones obtenidas al resolver una misma instancia con soluciones iniciales diferentes, es mucho menor que las variaciones en las soluciones que dan los otros dos algoritmos, esto es, la solución del algoritmo de recocido simulado es menos dependiente de la solución inicial proporcionada. En cambio, para el algoritmo estocástico y, más aún, para el algoritmo de búsqueda local, las soluciones dependen en gran medida de la solución con que se inicie. Este comportamiento se puede observar en la Tabla 4.9 en donde se muestran los resultados al ejecutar cada uno de los algoritmos 100 veces con soluciones iniciales generadas de manera aleatoria para cada uno de los problemas de Nugent.

Las pruebas computacionales con el problema de asignación cuadrática revelan un excelente comportamiento del algoritmo. Las soluciones subóptimas difieren en 1-2% de la mejor solución conocida. De hecho, si se ejecuta el procedimiento varias veces con diferentes valores iniciales, se obtienen en todos los casos la solución óptima o las mejores soluciones que se conocen (según sea el caso). Además de que la implantación del algoritmo, no requiere de gran capacidad de memoria y permite un tiempo razonable para obtener buenas soluciones en las computadoras personales.

	ALGORITMO							
	EXACTO		HEURISTICO		REC. SIM.		BUS. LOC.	
n	efic.	tiempo (seg.)	efic.	tiempo (seg.)	efic.	tiempo (seg.)	efic.	tiempo (seg.)
5	1.00	0.08	0.9891	0.08	0.9999	1.73	0.9747	0.13
6	1.00	0.25	0.9803	0.16	0.9982	2.25	0.9666	0.21
7	1.00	1.03	0.9767	0.28	0.9936	2.98	0.9676	0.29
8	1.00	3.98	0.9737	0.44	0.9899	3.79	0.9630	0.43
10	1.00	73.45	0.9684	1.01	0.9905	6.40	0.9629	0.79
12	1.00	1713.99	0.9674	2.03	0.9940	12.73	0.9512	1.07
15	—	—	0.9693	4.75	1.0000	15.21	0.9640	2.40
20	—	—	0.9512	14.55	1.0000	20.04	0.9518	5.64
25	—	—	0.9527	35.02	1.0000	28.62	0.9551	11.11
30	—	—	0.9543	71.80	1.0000	43.65	0.9580	20.96

Tabla 4.8: Comparación de los algoritmos para resolver el QAP con respecto al tiempo de ejecución y a su eficiencia.

	HEURÍSTICO			REC. SIM.			BUS. LOC.		
n	Mejor	Media	Peor	Mejor	Media	Peor	Mejor	Media	Peor
5	50	51.6	58	50	50.4	58	50	55.2	62
6	86	88.5	94	86	86.3	92	86	90.2	94
7	148	153.9	164	148	148.7	156	148	154.5	172
8	214	228	252	214	215.3	222	214	224.5	248
12	578	620.3	662	578	600.2	640	578	610.8	650
15	1188	1231	1294	1150	1172	1214	1150	1204	1290
20	2692	2749	2794	2570	2657	2728	2604	2680	2742
30	6488	6636	6766	6124	6333	6442	6242	6364	6496

Tabla 4.9: Comparación entre los algoritmos heurístico, recocido simulado y búsqueda local con respecto a la dependencia de la solución inicial.

4.4 Ejemplo: Sudoku

El Sudoku es un pasatiempo que en los últimos años se ha popularizado y para algunas personas se ha vuelto adictivo, quizá porque las reglas para resolverlo son muy simples. En esta sección se propone la solución al pasatiempo del Sudoku como un problema de optimización combinatoria y se resuelve utilizando el algoritmo de Recocido Simulado. Se presenta experiencia computacional en la solución de Sudokus que aparecen en periódicos de circulación nacional y algunas páginas de Internet con diferentes grados de dificultad que van desde fácil, media, difícil y muy difícil. Vea Sección 3.1.14

El objetivo del Sudoku es llenar una cuadrícula generalmente de 9×9 celdas (81 celdas en total) dividida en subcuadrículas de 3×3 con los números del 1 al 9, partiendo de algunas celdas previamente asignadas y siguiendo tres reglas muy sencillas. El pasatiempo aparece por primera vez alrededor de 1970 en Nueva York, Estados Unidos, en la revista de juegos y pasatiempos Dell con el nombre de "colocar el número". Posteriormente, en 1984, aparece este pasatiempo en Japón con el nombre que actualmente se conoce de Sudoku, donde "Su" significa número y "Doku" significa individualmente, en japonés. A partir de ahí, se populariza hacia todo el mundo. El juego tiene sus orígenes en la idea de cuadrados latinos que introduce el matemático Euler en 1783. En México hay muchas revistas de pasatiempos y periódicos de circulación nacional tales como El Universal, Reforma, etc., que incluyen Sudokus dentro de sus páginas.

4.4.1 El Problema del Sudoku

El objetivo del Sudoku es llenar una cuadrícula de $n^2 \times n^2$ celdas, dividida en n^2 subcuadrículas de $n \times n$ en donde se deben colocar los enteros del 1 a n^2 de acuerdo a las tres reglas, ya mencionadas Sección 3.1.14 pero por comodidad las repetimos aquí:

1. Las celdas de cada uno de los renglones deben contener los enteros de 1 a n^2 exactamente una vez.

2. Las celdas de cada una de las columnas deben contener los enteros de 1 a n^2 exactamente una vez.

1			7	3				
						2	5	
8								
				4				1
		5			6			
	2							
4							3	
					2	4		
	1		5					

Figura 4.5: Sudoku.

3. Cada una de las subcuadrículas debe contener los enteros de 1 a n^2 exactamente una vez.

Estas reglas tan simples son quizá lo que ha hecho que el juego se popularice y se haya hecho adictivo para algunas personas. Al entero n se le llama el orden del Sudoku. En la Figura 4.5 se da un ejemplo de un Sudoku de orden 3, donde se puede observar que existen algunas celdas prellenadas con los enteros del 1 al 3^2 y el jugador debe completar la cuadrícula siguiendo las tres reglas descritas anteriormente. Cada Sudoku, bien planteado, tiene solución válida única y no requiere usar técnicas de ensayo y error o adivinar sino que se resuelve empleando las tres reglas y por medio del "razonamiento".

El Sudoku de la Figura 4.5 tiene solución válida única y está catalogado entre los muy difíciles de resolver. Generalmente en los diarios de circulación nacional o internacional, así como, en revistas y otros medios impresos aparecen Sudokus de orden 3 y en algunos casos de orden 4 en donde aparece alguna clasificación que va desde fácil hasta muy difícil y tiene que ver con el número de celdas prellenadas y su colocación dentro de la cuadrícula. En el caso del Sudoku de la Figura 4.5, el número de celdas prellenadas es de 17, y aparentemente, es el número mínimo de celdas prellenadas que se pueden dar para Sudokus de orden 3, ya que con menos el Sudoku podría tener más de una solución válida.

Para ilustrar cómo se resuelve un Sudoku, considere la celda en donde se colocó el símbolo α en la Figura 4.6. Las celdas donde aparecen los 1 (en negritas) hacen que las celdas marcadas con $\times$ de la subcuadrícula donde

1			7	3				
						2	5	
8								
×	×	×		4				**1**
×	×	5			6			
×	2	α						
4							3	
					2	4		
	1		5					

Figura 4.6: Resolviendo el Sudoku.

aparece la letra α no pueda colocarse el entero 1, ya que violarían las reglas 1 y 2 anteriores, como las × invalidan todas las celdas, excepto donde aparece la letra α, es claro que en esta celda debe colocarse el símbolo 1. Por otro lado, considere la celda en donde se colocó el símbolo β, en la Figura 4.7. Las celdas donde aparecen los símbolos 4 (en negritas) hacen que las celdas marcadas con × de la subcuadrícula donde aparece la letra β no pueda colocarse el símbolo 4, ya que nuevamente violarían las reglas 1 y 2, como las × invalidan todas las celdas, excepto donde aparece la letra β, es claro que en esta celda debe colocarse el símbolo 4. Se sigue de la misma forma hasta completar todas las celdas vacías. En la Figura 4.8 se da la solución válida del Sudoku que aparece en la Figura 4.5.

Finalmente, Yato y Seta (2002)[115] muestran que el Sudoku es un problema NP-Completo, la prueba usa una reducción simple para el problema de cuadrados latinos, el cual ha sido probado que es NP-Completo por Colbourn (1984)[27].

4.4.2 Implementación de la Técnica de Recocido Simulado al Sudoku

En esta subsección se explica cómo se implementa el algoritmo de Recocido Simulado para resolver el problema del Sudoku. Se utilizará un Sudoku de 9×9 celdas con 9 subcuadrículas de 3×3, pero el procedimiento puede usarse para implementar cualquier Sudoku de orden n. Se representará el Sudoku con un arreglo de 9×9 en donde existen celdas previamente llenadas. A las celdas previamente llenadas se les llamarán *celdas fijas*. Por ejemplo en la

1			7	3				
						2	5	
8								
				4				1
		5			6			
	2	α						
4			$\times$	$\times$	$\times$		3	
			$\times$	$\times$	2	**4**		
	1		5	$\times$	β			

Figura 4.7: Resolviendo el Sudoku (continuación).

1	5	2	**7**	**3**	8	9	6	4
7	3	9	4	6	1	**2**	**5**	8
8	6	4	2	5	9	1	7	3
6	8	7	9	**4**	5	3	2	**1**
3	4	**5**	1	2	**6**	8	9	7
9	**2**	1	8	7	3	6	4	5
4	9	8	6	1	7	5	**3**	2
5	7	6	3	8	**2**	**4**	1	9
2	**1**	3	**5**	9	4	7	8	6

Figura 4.8: Solución válida al Sudoku de la Figura 4.5.

Figura 4.5, la celda (4,5) tiene el valor fijo igual a 4, o la celda (7,8) tiene el valor fijo de 3. A través del procedimiento, a las celdas fijas nunca se les cambiará el valor asignado.

Solución inicial

El objetivo del algoritmo que se desarrollará es obtener una solución válida al Sudoku, en donde solución válida significa un llenado del Sudoku donde se respeten los valores iniciales de las celdas fijas y se satisfagan las tres reglas ya mencionadas. En lo que sigue, definiremos una *solución factible* o simplemente *solución* de un Sudoku a cualquier llenado del Sudoku que cumpla con dos condiciones:

1. Respete los valores de las celdas fijas.
2. Cumpla con la regla 3.

Observe que esta definición de solución no implica que sea una solución válida del Sudoku porque no necesariamente cumple con las reglas 1 y 2.

La solución inicial se generará al azar, es decir, se genera una permutación aleatoria de los números del 1 al 9 no fijos en cada una de las subcuadrículas. Por ejemplo en la Figura 4.5, la primera subcuadrícula tiene dos celdas fijas la (1,1) y la (3,1) con los valores 1 y 8 respectivamente, por lo tanto, quedan libres los números 2, 3, 4, 5, 6, 7 y 9. Con estos números se genera una permutación aleatoria y se rellenan las celdas faltantes. Así que, si la permutación aleatoria de los números anteriores fuera 9, 5, 7, 6, 2, 3, 4, se llenaría la primera subcuadrícula como en la Figura 4.9.

Procediendo de la misma forma con las siguientes 8 subcuadrículas obtendríamos el llenado dado en la Figura 4.10 (en donde aparecen numeradas las filas y las columnas). La solución inicial dada en la Figura 4.10 respeta las celdas fijas dadas originalmente (en negritas) y la regla 3, pero no necesariamente respeta las reglas 1 y 2. Para esta solución inicial obtenida se puede observar que en la fila 1 no aparecen los valores 2 y 6 y se repiten los valores 5 y 7; por otro lado, en la columna 5, aparecen los valores 3 (tres veces), 4 y 8 (dos veces), y no aparecen los valores 1, 2, 5, 6. Lo mismo podemos decir del resto de las filas y columnas. Obviamente esta "solución inicial" no es una solución válida para el Sudoku ya que las reglas 1 y 2 no se cumplen.

1	9	5	**7**	**3**				
7	6	2				**2**	**5**	
8	3	4						
				4				**1**
		5			**6**			
	2							
4							**3**	
					2	**4**		
	1		**5**					

Figura 4.9: Llenado inicial de la primera subcuadrícula.

	1	2	3	4	5	6	7	8	9
1	**1**	9	5	**7**	**3**	5	8	4	7
2	7	6	2	2	8	6	**2**	**5**	9
3	**8**	3	4	1	9	4	3	6	1
4	6	1	3	1	**4**	2	5	4	**1**
5	8	4	**5**	5	7	**6**	9	3	2
6	9	**2**	7	8	3	9	7	8	6
7	**4**	5	8	1	8	6	9	**3**	7
8	2	7	3	9	4	**2**	**4**	1	6
9	9	**1**	6	**5**	3	7	5	2	8

Figura 4.10: Solución inicial obtenida por el procedimiento.

Costo asociado a una solución

Vamos a asociar un costo (o una penalización) a cada solución factible que refleje que tan cerca o lejos esté de resolver el Sudoku. Entre más violaciones tengamos de las reglas 1 y 2, la penalización será mayor. Se asocia una penalización a cada una de las filas y columnas. Definiremos la penalización asociada a una fila como el número de enteros faltantes en la fila de entre los enteros 1, 2,..., 9 y de la misma manera la penalización asociada a la columna es el número de enteros faltantes de entre los enteros 1, 2,..., 9 en la columna. Por ejemplo para la solución factible de la Figura 4.10 la penalización de la fila 1 es igual a 2 ya que no aparecen el 2 y el 6, la penalización asociada a la columna 5 es 4 ya que no aparecen los enteros 1, 2, 5 y 6. Las penalizaciones asociadas a la solución factible de la Figura 4.10 se dan en la Tabla 4.10.

El *costo total* de una solución factible la definiremos como la suma de las penalizaciones asociadas a las filas y columnas; para la solución factible de la Figura 4.10 su costo total es de 41 unidades. Es fácil observar que si obtenemos una solución factible con costo total igual a cero, esta solución factible debe ser una solución válida para el Sudoku. Por lo tanto, el objetivo es ir mejorando las soluciones factibles hasta encontrar una solución cuyo costo total sea igual a cero.

Fila	1	2	3	4	5	6	7	8	9
Penalización	2	3	3	3	1	3	1	2	1
Columna	1	2	3	4	5	6	7	8	9
Penalización	2	1	2	3	4	3	2	2	3

Tabla 4.10: Penalizaciones asociadas a la solución inicial.

Observe que hemos definido un problema de optimización combinatoria donde se tiene un espacio de soluciones (las soluciones factibles dadas por las condiciones 1 y 2) y cada solución tiene un costo total asociado. La solución válida del Sudoku se obtiene cuando se descubre una solución con costo total igual a cero. Si el Sudoku está bien planteado, debe de existir exactamente una solución con costo total igual a cero.

	1	2	3	4	5	6	7	8	9
1	**1**	9	5	**7**	**3**	5	8	4	7
2	7	6	2	2	8	6	**2**	**5**	9
3	**8**	3	4	1	9	4	3	6	1
4	6	1	3	1	**4**	2	5	4	**1**
5	8	4	**5**		7	**6**	9	3	2
6	9	**2**	7	8		9	7	8	6
7	**4**	5	8	1	8	6	9	**3**	7
8	2	7	3	9	4	**2**	**4**	1	6
9	9	**1**	6	**5**	3	7	5	2	8

Figura 4.11: Solución vecina de la inicial.

Soluciones vecinas

Dada una solución factible actual, se desea generar otra solución factible, a partir de la primera, que cambie marginalmente. Para esto definimos las *soluciones vecinas* como todas aquellas que difieren de la actual al hacer un intercambio de los valores de dos celdas no fijas en la misma subcuadrícula. Observe que con esta definición se sigue cumpliendo que la nueva solución es factible ya que al hacer este cambio se cumplen las condiciones 1 y 2 de la subsección 3.1. Por ejemplo, si se seleccionan las celdas (5,4) y (6,5), de la solución inicial de la Figura 4.10 y se intercambian sus valores, se obtiene la solución vecina dada en la Figura 4.11.

Actualización de la función de costo

Si tenemos una solución factible y a partir de ésta se genera una solución factible vecina, el costo total de esta nueva solución se puede cambiar, pero no es necesario recalcular las penalizaciones en todas las filas y columnas para obtener el costo total de la nueva solución. Existen dos casos, tanto por fila como por columna. Para el cálculo de las penalizaciones por fila: si las celdas que intercambian valores aparecen en la misma fila, la penalización por fila queda exactamente igual ya que el intercambio afecta a una sola fila y este intercambio no agrega ni elimina ningún entero de esa fila, los enteros faltantes, si es que los hay, seguirían siendo los mismos. Por otro lado, si se intercambian valores en dos filas diferentes, entonces la actualización de la penalización en la fila se puede hacer en dos pasos: primero, si el entero

que se elimina (se cambia por el otro) ya aparece en otra posición en la fila, quitarlo no hace que falte y por lo tanto no afecta a la penalización. en caso contrario, se aumenta en uno la penalización; segundo, si el entero que se agrega ya existe en otra posición en la fila, agregarlo no afecta la penalización de la fila, en caso contrario, la penalización disminuye en uno. La actualización por columna es exactamente igual que para la fila.

Por ejemplo para la actualización del costo de la solución factible vecina dada en la Figura 4.11. En la fila 5 primero se elimina el entero 5, pero como ya aparece este entero en la celda (5,3), no afecta su eliminación, por otro lado se agrega el entero 3 que ya aparece en la celda (5,8) y por lo tanto no afecta el valor de la penalización al agregarlo, en suma la penalización de la fila 5 en la nueva solución es igual a la de la solución inicial y su valor es 1. Para la fila 6, se hace el intercambio contrario, se elimina el valor 3 que no aparece en otro sitio de la fila y por lo tanto la penalización se aumenta en 1, por otro lado se agrega el entero 5 que no aparece en ningún otro lado de la fila y la penalización se disminuye en 1, finalmente la suma algebraica de ambas operaciones es $+1 - 1 = 0$ y la penalización de la fila 6 es igual a 3. Para el caso de las columnas: comencemos con la columna 4, se elimina el entero 5, como también aparece en la celda (9,4), esta eliminación no afecta la penalización, por otro lado, se agrega el entero 3 que no aparece en el resto de la columna así que la penalización disminuye en 1 y por lo tanto la suma algebraica de ambas operaciones es $0 - 1 = -1$, así que, la penalización de la columna 4 cambia a $3-1 = 2$. Para la columna 5, primero se elimina el entero 3 que aparece en las celdas (1,5) y (9,5) y por lo tanto la penalización queda igual al eliminar el entero 3, al agregar el entero 5 se observa que no existe en dicha columna y por lo tanto la penalización de la columna baja en 1, así que el efecto total es $0 - 1 = -1$ y por lo tanto la penalización de la columna 5, queda como $4 - 1 = 3$. Finalmente el costo total de la nueva solución vecina propuesta es 39. La nueva solución factible se considera mejor que la inicial, dado que tiene un costo menor, es decir, tiene menos violaciones a las reglas 1 y 2.

Mecanismo de generación

El número de soluciones vecinas depende del número de celdas fijas, entre más celdas fijas haya se tendrán menos soluciones vecinas y viceversa. Se requiere crear un mecanismo de generación de una solución vecina a partir de la solución actual. La forma de hacerlo es: primero se selecciona al

azar una celda no fija dentro de todas las celdas no fijas de la cuadrícula de 9×9; enseguida se selecciona al azar una celda no fija perteneciente a la subcuadrícula a la que pertenece la primera celda no fija seleccionada. Posteriormente se intercambian los valores enteros asignados a estas dos celdas seleccionadas.

Para crear una solución vecina de la solución actual dada en la Figura 4.10, se selecciona al azar una celda, es decir, seleccionamos al azar dos valores con distribución uniforme discreta entre 1 y 9, el primer valor se lo asignamos al número de fila y el segundo al número de columna. Por ejemplo, si obtenemos los valores 5 y 2 corresponden a la celda (5,2) que es una celda no fija (en caso que fuera una celda fija se intentaría nuevamente hasta obtener una celda no fija) esta celda corresponde a la subcuadrícula 4 por lo que la siguiente selección se haría seleccionando un número al azar con distribución uniforme entre 4, 5 y 6 y otro un número al azar con distribución uniforme entre 1, 2 y 3, nuevamente si está selección corresponde a una celda fija o a la celda ya seleccionada (la (5,2)), en este caso se seleccionaría otra pareja hasta obtener una celda no fija diferente a la (5,2); suponga que la celda seleccionada sea la (6,3), por lo que la solución vecina generada al azar sería aquella que resulta de intercambiar los enteros 4 y 7 en las celdas seleccionadas.

El parámetro de control

El valor inicial del parámetro de control c_0 se obtuvo realizando varias iteraciones con soluciones vecinas generadas a partir de una solución inicial aleatoria x_0. Se hicieron $NOFIJA^2$, iteraciones, donde el valor $NOFIJA$ es el número de celdas no fijas del Sudoku que se va a resolver. Sea $f(x)$ el costo asociado a la solución x, para este conjunto de soluciones se calculó el costo promedio $\overline{f}$ y la desviación estándar $\sigma_{\overline{f}}$. Se tomó el valor inicial del parámetro de control como: $c_0 = \sigma_{\overline{f}}$.

Los valores sucesivos del parámetro de control se obtuvieron mediante la ecuación geométrica $c_{k+1} = \alpha c_k = \alpha^{k+1} c_0$ con un valor de $\alpha = 0.95$, 0.97 y 0.99 de acuerdo a la dificultad del Sudoku. Para cada valor fijo del parámetro de control se realizaron $NOFIJA^2$ iteraciones.

4.4.3 Algoritmo de Recocido Simulado para el Sudoku

El pseudocódigo del algoritmo de Recocido Simulado desarrollado para el juego del Sudoku se da en la Figura 4.12. En las líneas 2-4 se toman los valores iniciales del parámetro de control c_0, k, *equilibrio*, *transiciones*, el valor α que se toma igual a 0.95, 0.97 o 0.99 según sea el grado de dificultad del Sudoku y la solución actual x_0. El ciclo interior (líneas 6-15) se ejecuta tantas veces como el número $NOFIJA^2$, donde se mantiene constante el parámetro de control. Si se alcanza una solución x_j que tenga un costo $f(x_j) = 0$, entonces se habrá obtenido la solución válida del Sudoku y el algoritmo termina (línea 8), en caso contrario se ejecuta totalmente el ciclo interior hasta obtener el equilibrio para el valor fijo c_k. El ciclo exterior (líneas 5-17) cambia el parámetro de control mediante la ecuación (2) en la línea 16 y controla el número de iteraciones hasta el punto de congelamiento (c_{final}) que se está definiendo como aquel en el que después de ejecutar un ciclo interior completo no hay cambios en la solución. Si el algoritmo termina con esta condición de congelamiento (línea 17) entonces no se habrá encontrado ninguna solución válida al Sudoku. En este caso se puede nuevamente "calentar" o recalentar, es decir, tomar nuevamente el valor inicial c_0 y la solución inicial igual a la solución actual y regresar a la línea 4 del algoritmo. Si después de seguir el procedimiento anterior, en reiteradas ocasiones, no se encuentra una solución válida al Sudoku se podría sospechar que el Sudoku que se está tratando de resolver no tiene solución válida.

4.4.4 Resultados

Se tomaron Sudokus que aparecen en periódicos de circulación nacional tales como Reforma, El Universal, etc., y en portales de Internet como en http://www.gamehouse.com/onlinegames/, http://www.sudoweb.com/, http://Sudoku.-zeit.de/, etc., en donde se catalogan en niveles de dificultad como fácil, medio, difícil y muy difícil. Se corrieron 100 Sudokus de cada nivel. Los resultados obtenidos aparecen en la Tabla 4.11 en donde se observa que el costo promedio para sukokus de nivel fácil es mucho menor (en diez unidades) que Sudokus de nivel medio hacia arriba ya que el número de celdas fijas es mucho mayor que los demás y por lo tanto el número de errores al asignar valores aleatoriamente al resto de las celdas no crece mucho como en el caso de los otros niveles que no varía mucho el número de celdas fijas (en promedio entre 22 y 25). También se puede observar que la variación

```
1  Inicio
2  x_i ← x_0
3  equilibrio ← NOFIJA^2, α ← 0.95, 0.97 o 0.99
4  transiciones ← 0,  c_0 ← σ_f̄,  k ← 0
5  Hacer
6      Hacer hasta transiciones = equilibrio
7        Seleccione al azar x_j ∈ V(x_i)
8              Si f(x_j) = 0 entonces SALIR (solucion valida del sudoku)
9              Si f(x_j) < f(x_i) entonces
10                   x_j ← x_i
11             Otro caso
12                   Si exp((f(x_j) − f(x_i))/c_k) ≥ random [0, 1) entonces
13                         x_j ← x_i
14       transiciones = transiciones + 1
15     Continúa
16   c_{k+1} ← α × c_k,  k ← k + 1,  transiciones ← 0
17 Hasta (No haya cambio de solucion)
18 Fin
```

Figura 4.12: Algoritmo de Recocido Simulado para el Sudoku.

del número de celdas fijas entre los niveles medio, difícil y muy difícil no es mucho. Las desviaciones estándar promedio de los costos entre los diferentes niveles de dificultad no es muy significativa ya que va de alrededor de 9 a 10 unidades de manera creciente con respecto al grado de dificultad. Para todas las corridas primero se calcula la desviación estándar de los costos de cada Sudoku y este valor se le asignó al valor inicial del parámetro de control. Los tiempos promedio de solución entre los diferentes niveles sí varía significativamente: de un tiempo promedio de 0.011 segundos, para el nivel fácil; se duplica en el nivel medio (.024 segundos), y se vuelve a duplicar el tiempo entre el nivel medio y difícil. La diferencia ya es más significativa entre el nivel difícil y muy difícil que se triplica. También se probaron varios valores para el parámetro α para los diferentes niveles obteniéndose valores de α idóneos para cada grado de dificultad. Para fácil y medio fue suficiente tomar el valor de $\alpha = 0.95$. Para el caso de nivel de dificultad difícil se tomó $\alpha = 0.97$ y para el nivel de dificultad muy difícil se tomó $\alpha = 0.99$. Con estos valores se garantizó la obtención de una solución válida a todos los Sudokus resueltos.

Nivel	Costo Prom.	Desv. Estan.	Celdas Fijas Prom.	Tiempo Prom. (seg)	No. Iter. c_k fijo	α
Fácil	36.14	8.95	40.85	.01125	1612	0.95
Medio	45.78	9.51	25.05	.02465	3130	0.95
Difícil	46.55	9.70	22.65	.04445	3404	0.97
Muy Difícil	46.46	9.96	22.85	.13350	3381	0.99

Tabla 4.11: Estadísticas obtenidas en la solución de los Sudokus.

4.4.5 Conclusiones

En esta sección se da una descripción del Sudoku y se plantea como un problema de optimización combinatoria en donde se define un espacio de soluciones factibles y a cada solución se le asocia un costo. Se desarrolla un algoritmo de Recocido Simulado para resolver el Sudoku, la solución válida al Sudoku es aquella cuyo valor objetivo es igual a cero. Se implementó el algoritmo y se corrieron instancias obtenidas de periódicos, revistas y algunas páginas de Internet cuyo grado de dificultad iba de fácil a muy difícil, obteniéndose siempre la solución válida en un tiempo muy pequeño.

4.5 Ejercicios

1. Suponga que la función objetivo es el número de unos en una cadena de longitud 4 de ceros y unos; se desea obtener el valor mínimo de esta función por medio de recocido simulado, defina para este problema:

   (a) La estructura de vecindades.

   (b) Un mecanismo de generación.

   (c) Haga cuatro iteraciones del algoritmo, tomando el valor de $c_0 = 0.5$ y solución inicial igual a (1,0,0,1).

2.

Para cada uno de los ejercicios del 2-10, realice las siguientes actividades:

   (a) Defina una estructura de vecindades y especifique un programa de enfriamiento.

   (b) Modifique los parámetros en el inciso (a) para optimizar sus resultados.

   (c) De los análisis hechos en los incisos (a) y (b). ¿Cuáles son los parámetros que usaría para resolver el problema? Justifique su respuesta.

   (d) Haga un reporte con todos sus análisis y resultados.

2. El problema de conjunto independiente, visto en la sección 3.1.4.

3. El problema de coloración mínima, visto en la sección 3.1.5.

4. El problema de empaquetamiento, visto en la sección 3.1.6.

5. El problema de cubrimiento por vértices, visto en la sección 3.1.7.

6. El problema del agente viajero visto en la sección 3.1.8.

7. El problema de clan máximo, visto en la sección 3.1.9.

8. El problema de corte máximo, visto en la sección 3.1.10.

9. El problema de satisfacibilidad, visto en la sección 3.1.11.

10. El problema de la mochila, visto en la sección 3.1.12.

11. Considere el problema de programación entera:

$$\begin{aligned} \text{Max } z &= 6x_1 + 8x_2 \\ \text{Sujeto a}: 3x_1 + 4x_2 &\leq 12 \\ 5x_1 + 2x_2 &\leq 10 \\ x_1 \geq 0,\ x_2 &\geq 0 \ \text{ y enteros} \end{aligned}$$

    (a) Resuelva el problema gráficamente.

    (b) Use recocido simulado para encontrar la solución óptima.

12. Un aserradero recibe troncos con una longitud de 20 pies cada uno y se cortan en longitudes más pequeñas para su venta a algunas compañías manufactureras. El aserradero tiene órdenes de compra para las siguientes longitudes: $l_1 = 3$ pies, $l_2 = 7$ pies, $l_3 = 11$ pies y $l_4 = 16$ pies. La compañía actualmente tiene un inventario de 2,000 troncos, cada uno, de 20 pies de longitud. Suponga que el aserradero actualmente tiene suficientes órdenes, de modo que su problema es determinar el patrón de corte que maximice sus beneficios. El beneficio que obtiene por cada uno de los cortes en longitud más pequeña es la siguiente:

Longitud (pies)	3	7	11	16
Beneficio ($)	1	3	5	8

    cualquier patrón de corte es permisible con tal que la longitud no exceda 20 pies, es decir,

$$3d_1 + 7d_2 + 11d_3 + 16d_4 \leq 20$$

    donde d_i es el número de piezas cortadas a la longitud $l_i, i = 1, 2, 3, 4$. Resuelva este problema usando el algoritmo de recocido simulado.

13. Una cadena de tiendas de deportes tiene tres tiendas en una ciudad. Un fabricante de pelotas de tenis le ofrece al director de la cadena de tiendas, 500 docenas de pelotas de tenis a un precio muy bajo. Las tres tiendas tienen una muy buena venta de artículos para tenis, por lo que el director decide comprar las pelotas. Así, el director ahora tiene el

problema de determinar que cantidad de pelotas asignará a cada una de las tiendas. La siguiente tabla muestra el beneficio esperado por asignar 100, 200, 300, 400 o 500 docenas de pelotas a cada tienda

	Número de docenas de pelotas de tenis				
Tienda	100	200	300	400	500
1	\$600	\$1100	\$1550	\$1700	\$1800
2	500	1200	1700	2000	2100
3	550	1100	1500	1850	1950

Suponga que los lotes no pueden ser separados en tamaños menores a 100 docenas.

Use recocido simulado para determinar la asignación óptima de las docenas de pelotas a las tres tiendas de deportes.

14. Una compañía contrata ocho nuevos empleados y debe determinar donde colocarlos. Existen cuatro actividades. La compañía ha preparado la siguiente tabla, que da el beneficio estimado para cada actividad en función del número de nuevos empleados que le sean asignados

	Número de nuevos empleados								
Actividades	0	1	2	3	4	5	6	7	8
1	22	30	37	44	49	54	58	60	61
2	30	40	48	55	59	62	64	66	67
3	46	52	56	59	62	65	67	68	69
4	5	22	36	48	52	55	58	60	61

(a) Use recocido simulado para determinar la asignación óptima de los nuevos empleados a las actividades.

(b) Suponga que únicamente seis nuevos empleados fueron contratados. ¿A qué actividades deben asignarse estos empleados?

15 Una empresa tiene a su disposición cinco proyectos para los próximos tres años. El valor presente de: los ingresos esperados, los gastos anuales y la disponibilidad de recursos disponibles durante estos años, de cada proyecto, se describen en la siguiente tabla:

Proyecto	Gastos anuales (millones)			Ingresos (millones)
	Año 1	Año 2	Año 3	
1	5	1	8	20
2	4	7	10	40
3	3	9	2	20
4	7	4	1	15
5	8	6	10	30
Recursos disponibles:	25	25	25	

El gerente desea seleccionar los proyectos que maximicen el valor presente neto de los ingresos de la empresa. Formule el modelo como un modelo de optimización combinatoria y resuélvalo usando recocido simulado.

16. Use el algoritmo de recocido simulado para resolver el problema del agente viajero con la matriz de costos no simétrica que aparece abajo.

a $j =$	1	2	3	4	5	6	7
de $i = 1$	×	13	9	19	22	3	7
2	6	×	7	8	39	4	8
3	18	3	×	5	18	4	3
4	4	16	6	×	5	13	16
5	14	22	99	17	×	8	10
6	3	21	7	3	10	×	9
7	11	19	8	19	8	3	×

17. En una fábrica textil hay un telar especial que fabrica una gran variedad de tejidos de un tipo especial. El primer día del mes la fábrica tiene órdenes de 7 trabajos que pueden procesarse en éste, para $i = 1, ..., 7$, p_i, d_i, r_i son respectivamente el tiempo de proceso en días, día de entrega y el beneficio del trabajo i. Estos datos se dan en la siguiente tabla:

i	1	2	3	4	5	6	7
p_i	9	10	12	5	11	8	13
d_i	4	13	15	8	20	30	30
r_i	90	130	85	35	77	68	100

Si el trabajo i se acepta, éste se tiene que entregar en el día de entrega para ese trabajo (donde d_i es el número de días contados a partir del

primer día del mes). Los trabajos son independientes y el telar puede procesar solamente un trabajo a la vez.

(a) Formular el problema de seleccionar los trabajos que serán aceptados maximizando el beneficio total sujeto a la restricción que todos los trabajos aceptados deben completarse en su respectiva fecha de entrega.

(b) Desarrolle un algoritmo de recocido simulado para obtener una buena solución a este problema.

Capítulo 5

Búsqueda Tabú

Muchos métodos para problemas combinatorios consisten de dos fases: construcción y mejoramiento. A continuación, se da una introducción sobre el método de la búsqueda tabú basada en [47], [48], [49] entre otros, y su implantación en computadora mediante ejemplos numéricos.

5.1 Introducción

Los intentos por tratar con el problema de la explosión combinatoria han encontrado muchos obstáculos. Por ejemplo, no es suficiente contar con un *conocimiento experto* para manejarlos de manera efectiva, de igual manera no es suficiente confiar en el poder computacional de alta velocidad de las super computadoras. Algunos problemas clásicos donde la explosión combinatoria prevalece, muestran que un intento por generar todas las alternativas relevantes por computadora no es una tarea factible.

Al respecto Glover (1989) expone: "la clave para tratar con tales problemas es ir un paso más allá de la aplicación directa de la destreza y del conocimiento del experto, y generar recursos para un procedimiento especial (o marco) el cual monitorea y dirige el uso de esta habilidad y conocimiento. Careciendo de tal procedimiento, las reglas expertas se pueden *empantanar*, permitiendo un punto donde ninguna mejora puede percibirse, a menos de que existan alternativas superiores".

Algunos problemas de optimización combinatoria como por ejemplo, el problema de programación lineal, el problema de transporte, el problema

de asignación; existen algoritmos rápidos y eficientes, pero en la mayoría de problemas de optimización combinatoria, no los hay. Como por ejemplo el problema de asignación cuadrático, el problema de localización de plantas, el problema del agente viajero, etc.

Desde hace varias décadas han sido atacados por algoritmos desarrollados especialmente para el problema específico y usando una diversidad de técnicas tales como planos de corte, ramificación y acotamiento, enumeración implícita, relajación Lagrangeana, partición de Benders, etcétera, o por combinaciones de las técnicas antes mencionadas; sin embargo, no pueden resolverse de manera exacta usando tiempo y espacio de computadora razonable, aún cuando se tenga sólo un número moderado de variables. En la actualidad, la investigación se ha dirigido hacia el diseño de buenas heurísticas, es decir, algoritmos eficientes con respecto al tiempo de cómputo y al espacio de memoria, y con cierta verosimilitud de entregar una solución *buena* esto es, relativamente cercana a la óptima mediante el examen de sólo un pequeño subconjunto de soluciones del número total.

El principal problema de algunos algoritmos estocásticos es la dificultad de escapar de la optimalidad local y en algunas otras veces conocido como agujero-negro o valle profundo (ver Figuras 5.1 y 5.7); es decir, donde se mantienen una serie de soluciones suboptimales y en muchas ocasiones se llega a un ciclado dentro de este conjunto subotimal de la región factible, estando la solución optimal en otra región de factibilidad. Lo anterior, ha propiciado que el enfoque de la inteligencia artificial haya revivido como solución de problemas que requieren de la búsqueda heurística. Recientemente varias aproximaciones han surgido del manejo de problemas de decisión complejos, como son: *algoritmos genéticos, redes neuronales, recocido simulado, búsqueda tabú, análisis de objetivos y búsqueda dispersa*, entre otros. Cabe mencionar que estas heurísticas y metaheurísticas siguen teniendo gran aceptación para proporcionar metodologías en lo que ahora se denomina *inteligencia artificial.*

El propósito de este capítulo es presentar un algoritmo altamente eficiente en la solución de problemas de optimización combinatoria denominado *Búsqueda Tabú.* Algunos elementos de La Búsqueda Tabú son discutidos en un problema de calendarización. Otro ejemplo que se tratará es el de "recolección de basura sólida en plataformas petroleras". La Búsqueda Tabú (BT) es un procedimiento estocástico utilizado para resolver problemas de

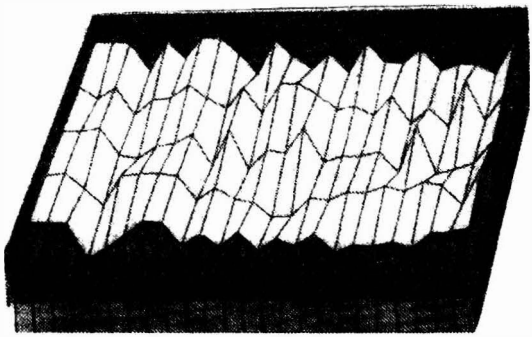


Figura 5.1: Gráfica del ejemplo de calendarización.

optimización de gran escala, dicho procedimiento estocástico está diseñado para guiar a otros métodos (o a procesos componentes) para escapar de la optimalidad local. La BT ha obtenido soluciones optimales o cercanas de la óptima en una amplia variedad de problemas clásicos.

La BT, junto con la técnica del recocido simulado y los algoritmos genéticos, han sido singularmente calificados ya desde hace tiempo por el *Committee on Next Decade of Operations Research* (Condor [1988]) como "extremadamente promisorios" para el tratamiento futuro de aplicaciones prácticas, aunque han aparecido otras técnicas como: el Grasp, PSO (*particle swam optimization*, en español *optimización por enjambre de partículas*), *Scatter Search* (búsqueda dispersa), método hormiga, etcétera.

La BT tiene como antecedentes a varios métodos diseñados para cruzar fronteras de factibilidad o de optimalidad local; sistemáticamente impone y relaja restricciones para permitir la exploración de regiones de otra manera prohibidas. Ejemplos iniciales de tales procedimientos incluyen heurísticas basadas sobre métodos de restricciones subrogadas y aproximaciones de planos de corte que violan de manera sistemática las condiciones de factibilidad.

En general, se puede decir que la filosofía de la BT está basada en manejar y explotar una colección de principios para resolver problemas de manera inteligente y tiene como elemento principal el uso de memoria flexible. Desde cierto punto de vista, el uso de memoria flexible involucra el proceso dual de crear y explotar estructuras para tomar ventaja de la *historia*, por lo que, se combinan las actividades de adquisición y mejoramiento de la información.

La BT se cimenta en tres puntos principales (c.f. [47], [48], [49]):

1. El uso de estructuras de memoria basadas en atributos diseñados para permitir criterios de evaluación e información de búsqueda histórica. la cual se explota más a fondo que las estructuras de memoria rígida (como en ramificación y acotamiento) o por sistemas de pérdida de memoria (como recocido simulado y otros métodos aleatorizados).

2. Un mecanismo asociado de control, mediante el empleo de estructuras de memoria, basado en el interjuego entre las condiciones que restringen y liberan al proceso de búsqueda (envuelto en las restricciones tabú y el criterio de aspiración).

3. La incorporación de funciones de memoria de diferentes lapsos, desde término corto hasta de término largo, para implantar estrategias que refuercen la combinación de movimientos y las características de solución que históricamente se han encontrado buenas, mientras que las estrategias de diversificación manejan la búsqueda dentro de nuevas regiones.

Las estructuras de memoria de la BT operan bajo cuatro dimensiones principales: pertenencia, frecuencia, calidad e influencia. El papel de estos elementos en la creación de procesos efectivos para resolver problemas es uno de los focos de atención de este trabajo. En este apartado, se desarrollan los elementos descritos para la elaboración de algoritmos originales altamente eficientes para atacar problemas clásicos considerados en la literatura como difíciles.

5.2 Algoritmo de Búsqueda Tabú

La BT es un procedimiento estocástico de "alto nivel" introducido y desarrollado en su forma actual por Fred Glover [47] y [48], el cual se utiliza con gran éxito para resolver problemas de optimización cuya característica principal es la de *escapar* de la optimalidad local.

> "La filosofía de la BT es la de manejar y explotar una colección de principios para resolver problemas de manera inteligente. Uno de los elementos fundamentales de la BT es el uso de la memoria flexible, desde el punto de vista de la BT, la memoria flexible envuelve el proceso dual de crear y explotar estructuras para tomar

ventaja mediante la combinación de actividades de adquisición, evaluación y mejoramiento de la información de manera histórica" (Glover y Laguna [49]).

En términos, generales el método de BT puede esbozarse de la siguiente manera:

Se desea mover paso a paso desde una solución factible inicial de un problema de optimización combinatoria hacia una solución que proporcione el valor mínimo de la función objetivo C. Para esto, se puede representar a cada solución por medio de un punto s (en algún espacio se define una vecindad $N(s)$ (ver Figura 5.2 para cada punto s como un conjunto de soluciones adyacentes a la solución s.

El paso básico del procedimiento consiste en empezar desde un punto factible s y generar un conjunto de soluciones en $N(s)$, de éstas se elige la mejor s^* y se posiciona en este nuevo punto ya sea que $C(s^*)$ tenga o no mejor valor que $C(s)$.

Hasta este punto se está cercano a las técnicas de mejoramiento local a excepción del hecho de que se puede mover a una solución peor s^* desde s.

La característica importante de la BT es, precisamente, la construcción de una lista tabú T de movimientos: aquellos movimientos que **no** son permitidos (movimientos tabú, véase Figura 5.2) en la presente iteración. La razón de esta lista es la de excluir los movimientos que nos pueden regresar a algún punto de una iteración anterior. Ahora bien, un movimiento permanece como tabú, sólo durante un cierto número de iteraciones, de forma que se tiene que T es una lista cíclica donde para cada movimiento $s \rightarrow s^*$ el movimiento opuesto $s^* \rightarrow s$ se adiciona al final de T donde el movimiento más viejo en T se elimina.

Las condiciones tabú tienen la meta de prevenir ciclos e inducir la exploración de nuevas regiones. La necesidad del significado de eliminar ciclos se debe a que, al moverse desde un óptimo local, una elección irrestricta de movimientos (persiguiendo aquellos con evaluaciones altas) permite igualmente regresarse al mismo óptimo local (véase Figura 5.3).

Hay que apuntar; sin embargo, que la eliminación de ciclos no es la última meta en el proceso de búsqueda. En algunos casos, una buena trayectoria de

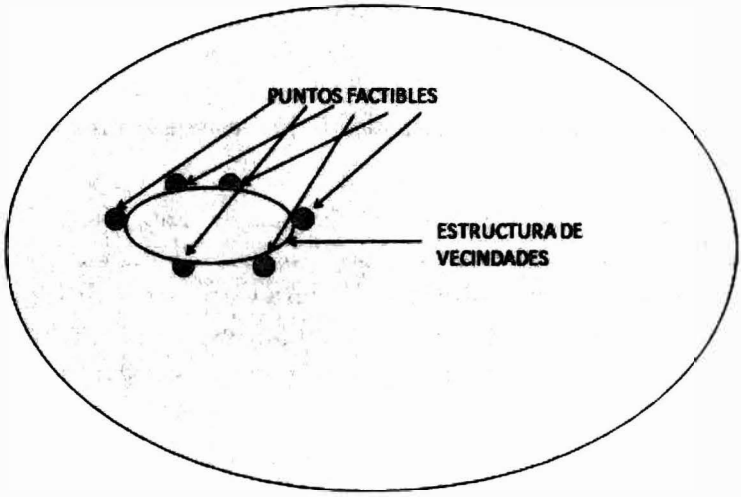


Figura 5.2: Estructura básica de vecindad.

búsqueda resultará al revisitar una solución encontrada anteriormente. El objetivo es el de continuar estimulando el descubrimiento de nuevas soluciones de alta calidad como se verá más adelante.

Ahora bien, las restricciones tabú no son inviolables bajo toda circunstancia. Cuando un movimiento tabú proporciona una solución mejor que cualquier otra previamente encontrada, su clasificación tabú puede eliminarse. La condición que permite dicha eliminación se llama *criterio de aspiración.*

Es así como las restricciones tabú y el criterio de aspiración de la BT, juegan un papel dual en la restricción y guía del proceso de búsqueda. Las restricciones tabú, permiten que un movimiento sea admisible si no está clasificado como tabú, mientras que si el criterio de aspiración se satisface, permite que un movimiento sea admisible aunque este clasificado como tabú.

La BT en una forma simple descubre dos de sus elementos claves: La de restringir la búsqueda mediante la clasificación de ciertos movimientos como prohibidos (es decir, tabú) y el de liberar la búsqueda mediante una función de memoria de término corto que proporciona una *estrategia de olvido.* Tres

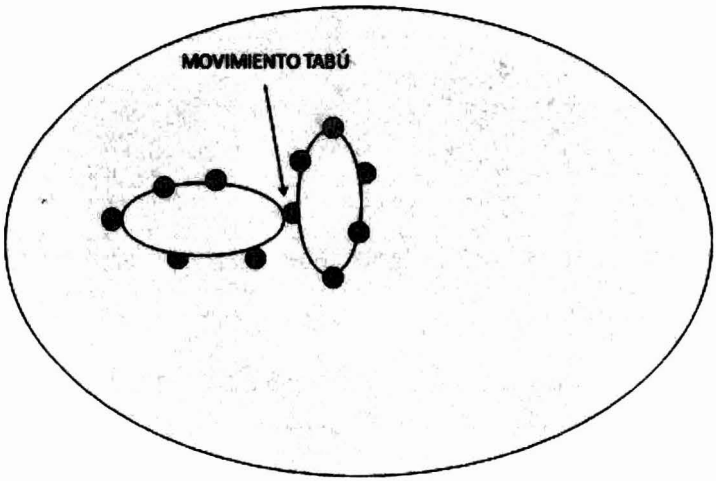


Figura 5.3: Representación gráfica de un punto tabú.

aspectos merecen énfasis:

1. El uso de T proporciona la "búsqueda restringida" de elementos de la aproximación y por lo tanto las soluciones generadas dependen críticamente de la composición de T y de la manera como se actualiza.

2. El método no hace referencia a la condición de optimalidad local, excepto implícitamente cuando un óptimo local mejora sobre la mejor solución encontrada previamente.

3. En cada paso se elige al *mejor* movimiento.

Para problemas grandes, donde las vecindades pueden tener muchos elementos, o para problemas donde esos elementos son muy costosos para examinar, es de importancia el aislar a un subconjunto de la vecindad y examinar este conjunto en vez de la vecindad completa. Esto puede realizarse en etapas, permitiendo al subconjunto de candidatos expandirse si los niveles de aspiración no se encuentran.

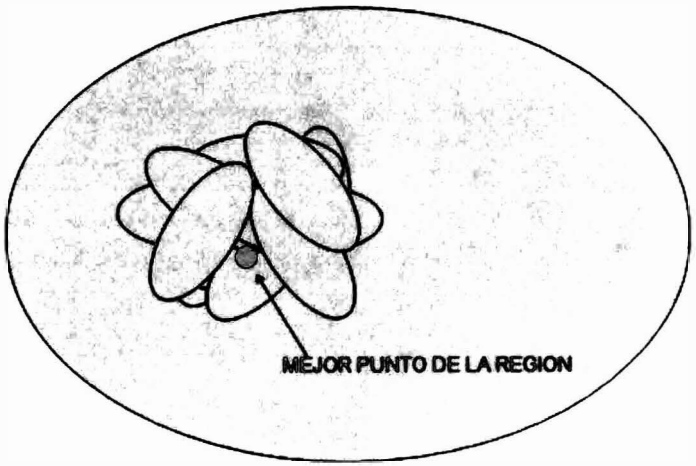


Figura 5.4: Proceso de intensificación regional.

Muchos problemas de optimización combinatoria tienen como soluciones factibles, las permutaciones π de un conjunto de objetos. Para explicar algunos conceptos de la BT, en lo que sigue, vamos a suponer que se está manejando como soluciones factibles permutaciones π de un conjunto de objetos y un movimiento de s a s^* consiste en obtener la permutación π^* a partir de la permutación π.

La selección preliminar de la clase de movimientos a tomarse dentro de la BT consiste en el cambio común por pares; es decir, se intercambia la posición de dos artículos para transformar una permutación a otra. Suponga que dada una permutación el objeto $\pi(i)$ precede, pero no necesariamente es adyacente al objeto $\pi(j)$. Un *movimiento de intercambio* es un rearreglo de los objetos $\pi(i)$ y $\pi(j)$ de forma tal que el objeto $\pi(i)$ se mueve a la posición j y el objeto $\pi(j)$ se mueve a la posición i. El *valor del movimiento* es la diferencia entre el valor de la función objetivo después del movimiento, $F(\pi^*)$, y el valor de la función objetivo antes del movimiento, $F(\pi)$, es decir:

$$valor_{movimiento} = F(\pi*) - F(\pi)$$

El valor de movimiento por lo general proporciona una base fundamental para evaluar la calidad de un movimiento, aunque otros criterios también son importantes como se verá más adelante.

Dada la solución inicial, el método realiza movimientos hasta que un tiempo de computadora (*tiempo_límite*) específico transcurra. El movimiento que se realiza en cierta iteración se encuentra revisando el valor de todos los movimientos *candidatos* para la solución actual. Un movimiento se considera que es un candidato si los trabajos a intercambiarse están dentro de una distancia específica (*número de posiciones*). Dado que se está minimizando, el mejor movimiento candidato es aquel que posee el menor valor algebraico. De manera más precisa, el mejor movimiento se selecciona del conjunto de movimientos *admisibles*. Un movimiento es admisible si es no tabú o sí su estatus tabú puede eliminarse por medio del criterio de aspiración. El mejor movimiento entonces se realiza y la estructura de datos tabú se actualiza.

El proceso fundamental mediante el que BT busca trascender la optimalidad local es el de introducir un mecanismo para hacer ciertos movimientos prohibidos. En la solución del problema, la preocupación principal es la de crear un estatus tabú que prevenga que algún movimiento se invierta bajo la jurisdicción de la memoria de término corto, la cual se escoge para que el problema tenga un número específico de movimientos futuros; es decir, la memoria de término corto de la BT constituye una forma de exploración agresiva que busca realizar el mejor movimiento posible (vea Esquema 1), sujeto a requerir elecciones posibles para satisfacer ciertas restricciones (vea Esquema 2, de la Figura 5.6). Esas restricciones están diseñadas para prevenir el regreso o repetir ciertos movimientos.

El Esquema 1, dado en la tabla 5.5, nos presenta la forma de elección del mejor movimiento admisible, esto es, dado un movimiento que es candidato, se elige al mejor de todos ellos considerando que, si es tabú, debe de satisfacer el criterio de aspiración. En el Esquema 2 de la Figura 5.6, se continúa con el proceso de búsqueda hasta que un criterio de paro se satisface; por lo general, consiste de un número predeterminado de iteraciones.

Existen varias formas para crear las restricciones tabú, la Tabla 5.1 (Laguna *et.al.* (1990)) muestra una lista de posibles atributos de un intercambio de los objetos $\pi(i)$ y $\pi(j)$ donde $j > i$, y que corresponden a restricciones que pueden imponerse para prevenir movimientos inversos.

1.	$(\pi(i),\pi(j),i,j)$	Impide cualquier movimiento que resulte en una permutación donde el objeto $\pi(i)$ ocupe la posición i y el objeto $\pi(j)$ ocupe la posición j.
2.	$(\pi(i),i,\pi(j),j)$	Impide cualquier movimiento que resulte en una permutación donde cualquiera de los objetos ya sea $\pi(i)$ ocupe la posición i o el objeto $\pi(j)$ ocupe la posición j.
3.	$(\pi(i))$	Impide a $\pi(i)$ regresarse a la posición i.
4.	$(\pi(i),i)$	Impide a $\pi(i)$ regresarse a una posición k con $k \leq i$.
5.	$\pi(i)$	Impide a $\pi(i)$ moverse.
6.	$(\pi(i),\pi(j))$	Impide a $\pi(i)$ y $\pi(j)$ moverse.

Tabla 5.1: Restricciones tabú y atributos para movimientos de intercambio.

Otra manera de identificar atributos de un movimiento de intercambio es el de introducir información adicional, mediante no sólo hacer referencia de los elementos intercambiados, sino también de las posiciones ocupadas por esos elementos en el momento de su cambio. Se puede observar que las primeras restricciones van de menor a mayor en cuanto a que son restrictivas, pero esto no se puede afirmar de manera uniforme. Ahora bien, no existe una forma que se pueda decir que es la mejor, esto sólo se puede realizar mediante pruebas. En ocasiones es importante asegurar que las condiciones puedan manejar los procesos de solución de manera eficaz desde la vecindad actual.

La meta principal de las restricciones tabú es el permitir al método ir a puntos más allá de la optimalidad local mientras se permita la realización de movimientos de alta calidad en cada paso al mismo tiempo de que exista una negociación balanceada con respecto al esfuerzo computacional al examinar muestras muy grandes, por lo que, en ocasiones es deseable incrementar el porcentaje de movimientos posibles para que reciban una clasificación tabú. Esto se puede lograr ya sea mediante el aumento en la pertenencia tabú o mediante el cambio de la restricción tabú.

Además, se requiere de una estructura de datos para guardar el seguimiento de los movimientos que son clasificados como tabú y para liberar aquellos movimientos de su condición tabú cuando su pertenencia a la memoria de término corto expire. El acompañamiento de la memoria basada en la pertenencia junto con la memoria basada en la frecuencia adicionan un compo-

Sea π^* la permutación que proporciona el mejor valor de la función objetivo hasta el momento, entonces:

Para (todos los movimientos candidatos)

Si (estatus del movimiento $\neq$ de tabú **o**

F(π)+*valor_movimiento* < F(π^*)

Si (*valor_movimiento* < *mejor_valor_movimiento*)

mejor_valor_movimiento $\leftarrow$ *valor_movimiento*;

mejor_movimiento $\leftarrow$ *movimiento_actual*;

Ejecute *mejor_movimiento*;

Figura 5.5: Esquema 1 Selección del mejor candidato admisible.

nente que típicamente opera sobre un horizonte. El efecto de tal memoria se puede estipular por medio de que la BT mantenga una historia selectiva *H* de los estados encontrados durante la búsqueda, reemplazando la vecindad actual *N(s)* por una vecindad modificada que depende de este proceso histórico *N(H,s)*.

El último elemento en el procedimiento básico es el *criterio del nivel de aspiración*, cuyo propósito es el de permitir que *buenos* movimientos tabú se seleccionen si el nivel de aspiración se alcanza. El apropiado uso de tal criterio puede ser muy importante para posibilitar que un método de BT alcance sus mejores niveles de realización. Este criterio de aspiración (que puede ser estándar) es el que permite que el estatus tabú se elimine si una mejor solución que la alcanzada hasta el momento se puede obtener, por ejemplo, a un movimiento tabú se le permite ejecutarse si:

$$F(\pi) + valor - movimiento < F(\pi*)$$

Ahora bien, otros criterios de aspiración pueden también proporcionar efectividad para mejorar la búsqueda, como se verá más adelante. Una base para uno de esos criterios proviene de introducir el concepto de *influencia* (Glover y Laguna [1993]), la cual mide el grado de cambio inducido en la

estructura de solución o de factibilidad. La influencia por lo general se asocia con la idea de distancia del movimiento, por ejemplo, donde un movimiento de gran distancia se concibe como de mayor influencia.

Genere solución inicial;

Haga

Crear lista de candidatos {Ejecute *mejor_movimiento*};

Actualice condiciones de admisibilidad;

Mientras (Criterio de paro = Falso);

Termine globalmente o transfiera.

Figura 5.6: Esquema 2 Componente de memoria de término corto de la búsqueda tabú.

El método inicia con una solución heurística factible, que se guarda como la mejor encontrada. Un paso crítico, el cual envuelve la orientación agresiva de la memoria de término corto, es la elección del mejor candidato admisible. La función *mejor_movimiento* es la que identifica a un movimiento para el cual el valor del movimiento es el más pequeño. El dominio de la función es el conjunto de todos los movimientos admisibles. El *mejor_movimiento* no tiene que ser necesariamente uno que mejore. Primero, cada uno de los movimientos de la lista de candidatos se evalúa en turno.

Ahora bien, conforme la búsqueda progresa, la forma de la evaluación empleada por la BT llega a ser más adaptativa, incorporando referencias concernientes para la **intensificación** (véase Figura 5.3 y la **diversificación** regional de búsqueda. [1]

[1]Cabe aclarar que en las estrategias basadas en consideraciones de término corto, la clasificación tabú sirve para identificar elementos de la vecindad del movimiento actual; mientras que en las estrategias de término intermedio y largo, pueden no contener soluciones en esta vecindad, por lo general consisten en seleccionar soluciones **élites** [óptimos locales de alta calidad] encontrados en varios puntos en el proceso de solución. Dichas solu-

La esencia del método depende de cómo el registro de la historia H se define y se utiliza, y de cómo los candidatos y la función de evaluación se determinan.

Revisar el estatus tabú es el primer paso en la escena de la admisibilidad. Si el movimiento no es tabú, es inmediatamente aceptado como admisible; de otra forma, el criterio de aspiración da una oportunidad para eliminar el estatus tabú, proporcionando al movimiento una segunda oportunidad para clasificarse como admisible.

En algunos casos, si las restricciones tabú y el criterio de aspiración son suficientemente limitados, ninguno de los movimientos posibles, serán clasificados como admisible. Un movimiento *menos inadmisible* se salva para manipular tal posibilidad y se elige si no emergen alternativas admisibles.

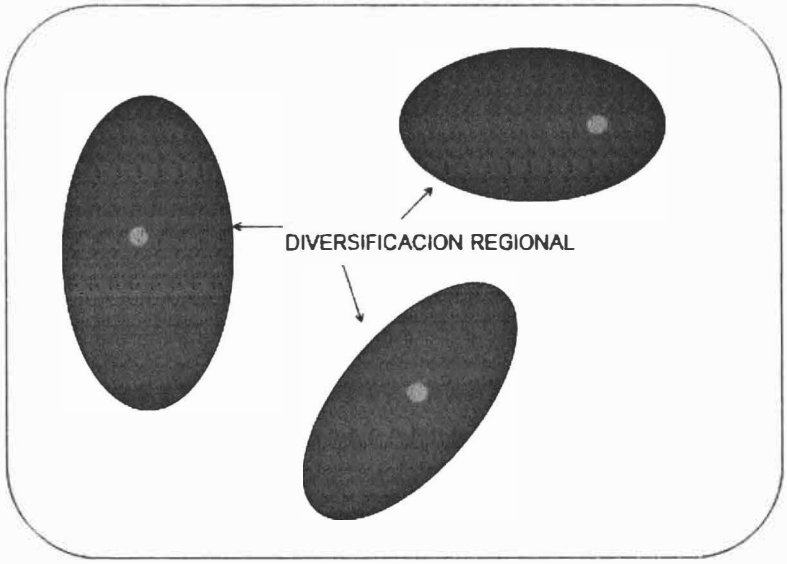


Figura 5.7: Diversificación regional.

La longitud de la lista tabú es un parámetro, si es demasiado pequeño el ciclado puede ocurrir, pero si es demasiado grande, restringirá bastante la

ciones élites se identifican como elementos de un conglomerado regional en las estrategias de intensificación de término intermedio, y como elementos de diferentes conglomerados en las estrategias de diversificación de término largo.

búsqueda para poder saltar *valles profundos* (*i.e.*, el mejor mínimo local) del espacio de valores de la función objetivo. Una faceta importante de la BT es la habilidad de localizar un rango robusto de longitudes de la lista tabú mediante pruebas empíricas preliminares para identificar (para una clase de problemas) los tipos de atributos y de restricciones tabú que se realizan de manera más efectiva. Acerca de esto, existe el uso de listas tabú múltiples, cada una desarrollada para un tipo particular de atributo.

Cuando diferentes tipos de atributos se manejan de esta forma, pueden estar dados con pesos distintos, dependiendo de su clasificación y edad, para determinar el estatus tabú de los movimientos que lo contienen.

En muchas aplicaciones, el componente de término corto por sí mismo ha producido soluciones superiores a aquellas encontradas mediante procedimientos alternativos y el uso de memoria de mayor término, en esos casos, se ha eludido. Ahora bien, la memoria de término intermedio y largo puede ser importante para obtener mejores resultados para problemas difíciles.

La memoria de término intermedio y largo opera primariamente como una base de las estrategias de intensificar y diversificar la búsqueda.

En términos generales, el método BT puede esbozarse de manera general en los siguientes pasos:

- Encontrar una solución inicial.
- Construir una estructura de vecindades. Se desea mover paso a paso desde un estado factible inicial de un problema de optimización combinatoria hacia un estado factible que proporcione el valor mínimo de la función objetivo F. Para esto, se puede representar a cada solución por medio de un punto m (en algún espacio) y se define un vecindario $\mathcal{V}(\Updownarrow)$ de cada punto m.
- Escoger el mejor vecino admisible. El paso básico del procedimiento consiste en empezar desde un punto factible m y generar un conjunto de soluciones en $\mathcal{V}(\Updownarrow)$; solo entonces se escoge al mejor vecino generado m^* y se posiciona en ese nuevo punto, ya sea que, $F(m^*)$ tenga o no mejor valor que $F(m)$. Hasta aquí, se está cercano a las técnicas de mejoramiento local a excepción del hecho de que se puede mover a una solución peor m^* desde m.
- Construir la tabla tabú y la función de memoria de largo plazo.

- La característica importante de la búsqueda tabú es, precisamente, la construcción de una lista tabú T de movimientos: aquellos movimientos que no son permitidos (movimientos tabú) en la iteración presente. La razón de esta lista es la de excluír los movimientos que nos pueden regresar a algún punto de una iteración anterior y evitar así el ciclado. Ahora bien, un movimiento permanece como tabú sólo durante un cierto número de iteraciones, de forma que se tiene que T es una lista cíclica donde para cada movimiento $m \rightarrow m^*$ el movimiento opuesto $m^* \rightarrow m$ se adiciona al final de T donde el movimiento más viejo en T se elimina. También se pueden construir tablas tabú de tipo dinámico como se verá posteriormente.
- Las condiciones tabú tienen la meta de prevenir ciclos e inducir la exploración de nuevas regiones. La necesidad del significado de eliminar ciclos se debe a que, al moverse desde un óptimo local, una elección irrestricta de movimientos (persiguiendo aquellos con evaluaciones altas) permite igualmente regresarse al mismo óptimo local.
- Hay que apuntar, sin embargo, que la eliminación de ciclos no es la última meta en el proceso de búsqueda. En algunos casos, una buena trayectoria de búsqueda resultará al revisitar una solución encontrada antes. El objetivo de manera amplia es el de continuar estimulando el descubrimiento de nuevas soluciones de alta calidad como se verá más adelante.
- Ahora bien, las restricciones tabú no son inviolables bajo cualquier circunstancia. Cuando un movimiento tabú proporciona una solución mejor que cualquier otra encontrada, su clasificación tabú puede eliminarse. La condición que permite dicha eliminación se llama *criterio de aspiración.*
- Es así como las restricciones tabú y el criterio de aspiración de la BT, juegan un papel dual en la restricción y guía del proceso de búsqueda.
- Las restricciones tabú permiten que un movimiento se observe como admisible si no se aplican, mientras que el criterio de aspiración permite que un movimiento se observe como admisible si se satisface.

- La búsqueda tabú en una forma simple descubre dos de sus elementos claves: La de restringir la búsqueda mediante la clasificación de ciertos movimientos como prohibidos (es decir, tabú) y el de liberar la búsqueda mediante una función de memoria de plazo corto que proporciona una *estrategia de olvido.*

- Actualizar las funciones de corto y largo plazo.
- Realizar una intensificación regional de búsqueda.
- Realizar una diversificación regional de búsqueda.

5.3 Ejemplo: Problema de Calendarización

A fin de describir diferentes aspectos de BT, se revisa el ejemplo en [15] y [72]. Considere el problema de calendarización de una máquina con costos de penalización por retraso y costos de actualización, ambos de tipo lineal. Como una aplicación considere que se tiene una máquina con las siguientes características: en el tiempo cero, N trabajos llegan a una máquina de capacidad continua. Cada trabajo i, $(i = 1, 2, \ldots, N)$ requiere de t_i unidades de tiempo en la máquina y tiene una penalización de retraso por cada unidad de tiempo de P_i unidades monetarias, a partir del tiempo cero; s_{ij} es el costo de actualización de calendarizar al trabajo j inmediatamente después del trabajo i. Dos trabajos falsos 0 y $N+1$ se incluyen en cada calendario, donde $t_0 = t_{N+1} = 0$ y $P_0 = P_{N+1} = 0$; los costos $s_{0,j}$ y $s_{i,N+1}$ se consideran como los costos de la puesta inicial y de limpieza respectivamente. Un calendario tiene la forma:

$$\pi = (0, \pi(1), \pi(2), \ldots, \pi(N), N+1),$$

donde $\pi(i)$ es el índice del trabajo en la posición i del calendario y el estado π es una permutación. El objetivo es el de minimizar la suma de los costos de actualización y de retraso para todos los trabajos. En términos matemáticos, se desea:

$$F(\pi) = D(\pi) + S(\pi) \rightarrow \min_{\pi} \qquad (P)$$

donde:

$$D(\pi) = \sum_{i=1}^{N} d_{\pi(i)} p_{\pi(i)},$$

$$S(\pi) = S_{0,\pi(1)} + \sum_{i=1}^{N-1} S_{\pi(i),\pi(i+1)} + S_{\pi(N),N+1},$$

$$d_{\pi(i)} = \sum_{j=1}^{i-1} t_{\pi(j)}, \quad i = 2, \ldots, N, \text{ y } d_{\pi(1)} = 0.$$

La clase de movimientos a tomarse dentro de la BT consiste en el cambio común por pares, es decir, se intercambia la posición de dos trabajos para transformar un calendario en otro. Suponga que dado un calendario el trabajo $\pi(i)$ precede, pero no necesariamente es adyacente al trabajo $\pi(j)$. Un *movimiento de intercambio* es un rearreglo de sólo los trabajos $\pi(i)$ y $\pi(j)$ de forma tal que el trabajo $\pi(i)$ se mueve a la posición j y el trabajo $\pi(j)$ se mueve a la posición i. El *valor del movimiento* es la diferencia entre el valor de la función objetivo después del movimiento, $F(\tilde{\pi})$, y el valor de la función objetivo antes del movimiento, $F(\pi)$; es decir:

$$\text{valor de movimiento} = F(\tilde{\pi}) - F(\pi).$$

Los valores de movimiento, por lo general, proporcionan una base fundamental para evaluar la calidad de un movimiento, aunque otros criterios también son importantes, como se verá más adelante.

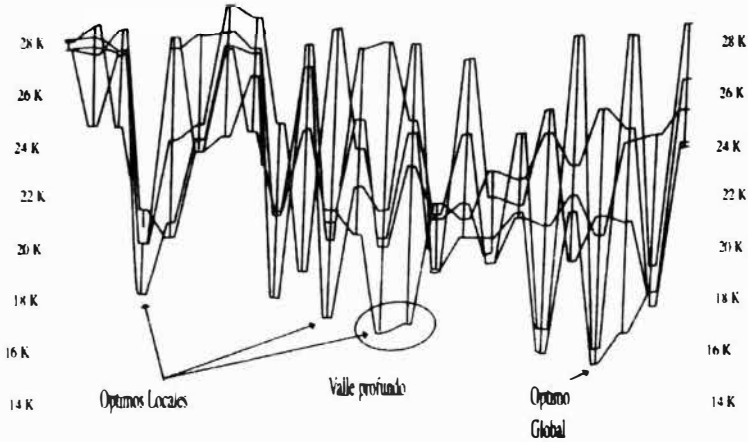


Figura 5.8: Representación gráfica de todos los posibles valores del ejemplo.

5.3.1 Algunas Observaciones para la Implementación

- El método inicia con un estado factible inicial, el cual se guarda como el mejor encontrado. El movimiento que se realiza en cada iteración es el de menor valor de todos los movimientos *candidatos* actuales. Un movimiento se considera que es un candidato si pertenece a la vecindad, en este ejemplo, se considera que se está en la vecindad, si los trabajos a intercambiarse están dentro de una distancia específica (número de posiciones). Dado que se está minimizando F, el mejor movimiento candidato es aquel que posee el menor valor de la función objetivo. De manera más precisa, el mejor movimiento se selecciona del conjunto de movimientos *admisibles*. Un movimiento es admisible si es no tabú o si su estado tabú puede eliminarse por medio del criterio de aspiración. El mejor movimiento entonces se realiza y la estructura de datos tabú se actualiza. El criterio de aspiración estándar, es el que permite que el estado tabú se elimine si una solución mejor que la alcanzada hasta el momento se puede obtener.

- La memoria de plazo corto de la BT constituye una forma de exploración agresiva que busca realizar el mejor movimiento posible, para satisfacer ciertas restricciones. Esas restricciones están diseñadas para prevenir el regresarse o la repetición de cierto número de veces de ciertos movimientos mediante la ejecución de atributos seleccionados de esos movimientos prohibidos (tabú).

  Otra manera de identificar atributos de un movimiento de intercambio es el de introducir información adicional, mediante no sólo hacer referencia de los elementos intercambiados sino también de las posiciones ocupadas por esos elementos en el momento de su cambio. Ahora bien, no existe una forma que se pueda decir cual es la mejor, sólo se puede identificar mediante pruebas.

  La meta principal de las restricciones tabú es el permitir al método ir a puntos más allá de la optimalidad local mientras se permita la realización de movimientos de alta calidad en cada paso, al mismo tiempo de que exista una negociación balanceada con respecto al esfuerzo computacional al examinar muestras muy grandes, por lo que, en ocaciones es deseable incrementar el porcentaje de movimientos posibles para que reciban una clasificación tabú. Esto se puede lograr ya sea mediante el aumento en la pertenencia tabú o mediante el cambio de la restricción

tabú.

- Se requiere además de una estructura de datos para guardar el seguimiento de los movimientos que son clasificados como tabú y para liberar aquellos movimientos de su condición tabú cuando su pertenencia a la memoria de plazo corto expire. El acompañamiento de la memoria basada en la pertenencia junto con la memoria basada en la frecuencia adicionan el efecto de que se puede llevar una historia selectiva de los estados encontrados durante la búsqueda, y reemplazando la vecindad actual por una vecindad modificada que depende de este proceso histórico.

- El último elemento en el procedimiento básico es el *criterio del nivel de aspiración*, cuyo propósito es el de permitir que movimientos tabú *buenos* se seleccionen si el nivel de aspiración se alcanza. El apropiado uso de tal criterio puede ser muy importante para posibilitar que un método de BT alcance sus mejores niveles de realización. El criterio de aspiración estándar, es el que permite que el estado tabú se elimine si una solución mejor que la alcanzada hasta el momento se puede obtener, por ejemplo, a un movimiento tabú se le permite ejecutarse si:

$$F(\pi) + valor_movimiento < F(\pi^*).$$

  Un paso crítico, el cual involucra la orientación agresiva de la memoria de corto plazo, es la elección del mejor candidato admisible. La función *mejor_movimiento* identifica a un movimiento para el cual el valor del movimiento es el más pequeño (para este ejemplo). El dominio de la función es el conjunto de todos los movimientos admisibles. El *mejor_movimiento* no tiene que ser necesariamente uno que mejore. Primero, cada uno de los movimientos de la lista de candidatos se evalúa en turno.

  Ahora bien, conforme la búsqueda progresa, la forma de la evaluación empleada por la búsqueda tabú llega a ser más adaptativa, incorporando referencias concernientes para la *intensificación* y la *diversificación* regional de búsqueda.

- Un criterio de paro utilizado en este tipo de métodos, es el de considerar un tiempo límite de cómputo prefijado y/o un número prefijado de iteraciones.

Trabajo i	Duración t_i	Penalización P_i	Radio t_i/P_i	Orden Natural
1	3	700	.004286	5
2	4	800	.005	4
3	1	100	.01	3
4	4	300	.0133	2
5	5	200	.025	1

Tabla 5.2: Condiciones del ejemplo.

S_{ij} Matriz de actualización de costos						
i/j	1	2	3	4	5	6
0	1100	600	1200	2000	1400	∞
1	∞	1300	700	1200	1100	1000
2	900	∞	1100	1300	600	1200
3	900	1000	∞	2000	700	1500
4	1000	700	800	∞	600	1200
5	1400	1300	1200	1300	∞	900

Tabla 5.3: Costos del ejemplo.

A continuación se proporciona un ejemplo numérico del problema de calendarización expuesto en la sección anterior, se mostrará cómo se puede llevar a cabo la implementación de algunos de los concepos vertidos en este capítulo, así como resolver algunos problemas que se presentan.

Se considerarán las siguientes tablas de datos dadas en [15]; para este ejemplo, los trabajos se indexan inicialmente de acuerdo al siguiente orden: $i < j$ implica que $\frac{t_i}{P_i} < \frac{t_j}{P_j}$.

En particular, para este ejemplo, se conoce la solución óptima que es: $F(\pi) = 13,500$ para $\pi = (0, 2, 1, 4, 3, 5, 6)$, recuérdese que los trabajos 0 y 6 son sólo trabajos falsos. En este ejemplo se considerará el criterio que impide cualquier movimiento que resulte en un calendario donde cualquiera de los trabajos ya sea a $\pi(i)$ ocupe la posición i o el trabajo $\pi(j)$ ocupe la posición j, además se tienen las tablas de condiciones 5.2 y de costos 5.3.

Las condiciones iniciales son:

- Punto Inicial π_0=(0,5,4,3,2,1,6).

- $F(\pi_0) = 26600$.
- Longitud_tabu = 7.
- Distancia máxima = 1.

Para iniciar nuestro proceso de búsquda tabú consideraremos como punto inicial el definido como orden natural, una longitud tabú fija igual a 7 así como una distancia de permutacuón igual a uno, por lo que se tendrán los datos anteriores.

Se presentarán las tablas de iteraciones, en donde se indicará el número de iteración del proceso, los calendarios generados y los correspondientes valores de la función objetivo, así como si el calendario propuesto es un movimiento tabú y el mejor movimiento admisible (aquél con valor objetivo menor o en su caso el menos peor) para continuar con el proceso.

En este ejemplo, las vecindades completas se examinan, entonces el mejor cambio se realiza, en este ejemplo, el que minimice la función objetivo y no sea movimiento tabú (o en el caso de que lo sea), para que sea admisible, debe de satisfacer el criterio de aspiración se considera aquél que satisface que el valor de la función objetivo mejora sobre todos los valores anteriormente encontrados.

La matriz tabú se construye al inicio del procedimiento, donde las filas de la matriz representan las posiciones y las columnas a los trabajos, se actualiza en cada iteración durante la fase de mejoramiento del algoritmo. Si un elememento (i, j) pertenece a esta matriz en una iteración dada, no se le permite realizar el cambio del trabajo i a la posición j, se actualiza en cada iteración. Recuérdese que es posible vencer la restricción tabú en el caso de que se satisfaga el criterio de aspiración

La matriz de frecuencias es la que lleva la *historia* del procedimiento y es la que se utiliza para la formación de la función de memoria de plazo largo, la cual permite la diversificación de la búsqueda; es decir, es posible el dirigir la búsqueda *más cercana* o *más alejada* de las regiones exploradas. A continuación se presentan algunas iteraciones, así como las tablas tabú y la función memoria de largo plazo, con respecto a la inicialización anteriormente indicada.

En la tabla 5.4, se puede observar que existen dos movimientos admisibles que nos proporcionan los mejores valores de la función objetivo, por lo que

Iteración	π	$F(\pi)$	Tabú = 1	Mejor Admisible
0	(0,5,4,3,2,1,6)	26600		*
1	(0,4,5,3,2,1,6)	26200		*
	(0,5,3,4,2,1,6)	27300		
	(0,5,4,2,3,1,6)	26200		*
	(0,5,4,3,1,2,6)	26700		

Tabla 5.4: Punto inicial y primera iteración.

Matriz tabú				
0	0	0	0	7
0	0	0	7	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

Matriz de Frecuencias				
0	0	0	1	0
0	0	0	0	1
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

Tabla 5.5: Lista tabú e historial de la primera iteración.

se puede escoger cualquiera de los dos. En este caso se elige el primero, por ejemplo, el movimiento que nos proporciona el calendario:

$$\pi = (0, 4, 5, 3, 2, 1, 6),$$

como punto inicial para la siguiente iteración.

Observe de la tabla 5.8 que en este caso el movimiento que nos proporciona el calendario $\pi = (0, 4, 3, 2, 5, 1, 6)$ es el movimiento que toma el valor objetivo más pequeño en la vecindad, por lo que se toma como punto inicial para la siguiente iteración.

En la cuarta iteración el único movimiento admisible es el que proporciona el calendario $\pi = (0, 4, 3, 2, 1, 5, 6)$ con un valor objetivo de $F(\pi) = 19,800$.

Note que, en la quinta iteración los movimientos que proporcionan los calendarios $\pi = (0, 4, 2, 3, 1, 5, 6)$ y $\pi = (0, 4, 3, 1, 2, 5, 6)$, son movimientos tabú, pero ambos satisfacen el criterio de aspiración, por lo que, se toma como punto inicial para la siguiente iteración el calendario con valor objetivo más pequeño en la vecindad.

Iteración	π	$F(\pi)$	Tabú = 1	Mejor Admisible
1	(0,4,5,3,2,1,6)	26200		
2	(0,5,4,3,2,1,6)	26600	1	
	(0,4,3,5,2,1,6)	25900		*
	(0,4,5,2,3,1,6)	26000		
	(0,4,5,3,1,2,6)	26300		

Tabla 5.6: Segunda iteración.

Matriz Tabú				
0	0	0	0	6
0	0	0	6	7
0	0	7	0	0
0	0	0	0	0
0	0	0	0	0

Matriz de Frecuencias				
0	0	0	1	0
0	0	1	0	1
0	0	0	0	1
0	0	0	0	0
0	0	0	0	0

Tabla 5.7: Lista tabú e historial de la segunda iteración.

Iteración	π	$F(\pi)$	Tabú = 1	Mejor Admisible
2	(0,4,3,5,2,1,6)	25900		*
3	(0,3,4,5,2,1,6)	26100	1	
	(0,4,5,3,2,1,6)	26200	1	
	(0,4,3,2,5,1,6)	22800		*
	(0,4,3,5,1,2,6)	26200		

Tabla 5.8: Tercera iteración.

Matriz Tabú				
0	0	0	0	5
0	0	0	5	6
0	0	6	0	7
0	7	0	0	0
0	0	0	0	0

Matriz de Frecuencias				
0	0	0	1	0
0	0	1	0	1
0	1	0	0	1
0	0	0	0	1
0	0	0	0	0

Tabla 5.9: Lista tabú e historial de la tercer iteración.

Iteración	π	$F(\pi)$	Tabú = 1	Mejor Admisible
3	(0,4,3,2,5,1,6)	22800		*
4	(0,3,4,2,5,1,6)	22800	1	
	(0,4,2,3,5,1,6)	22500	1	
	(0,4,3,5,2,1,6)	25900	1	
	(0,4,3,2,1,5,6)	19800		*

Tabla 5.10: Cuarta iteración.

Matriz Tabú				
0	0	0	0	4
0	0	0	4	5
0	0	5	0	6
0	6	0	0	7
7	0	0	0	0

Matriz de Frecuencias				
0	0	0	1	0
0	0	1	0	1
0	1	0	0	1
1	0	0	0	1
0	0	0	0	1

Tabla 5.11: Lista tabú e historial de la cuarta iteración.

Iteración	π	$F(\pi)$	Tabú = 1	Mejor Admisible
4	(0,4,3,2,1,5,6)	19800		*
5	(0,3,4,2,1,5,6)	19800	1	
	(0,4,2,3,1,5,6)	19400	1	
	(0,4,3,1,2,5,6)	19200	1	*
	(0,4,3,2,5,1,6)	22800	1	

Tabla 5.12: Quinta iteración.

Matriz Tabú				
0	0	0	0	3
0	0	0	3	4
0	7	4	0	5
7	5	0	0	6
6	0	0	0	0

Matriz de Frecuencias				
0	0	0	1	0
0	0	1	0	1
1	1	0	0	1
1	1	0	0	1
0	0	0	0	1

Tabla 5.13: Lista tabú e historial de la quinta iteración.

Iteración	π	$F(\pi)$	Tabú = 1	Mejor Admisible
5	(0,4,3,1,2,5,6)	19200		*
6	(0,3,4,1,2,5,6)	19600	1	
	(0,4,1,3,2,5,6)	18500	1	*
	(0,4,3,2,1,5,6)	19800	1	
	(0,4,3,1,5,2,6)	23200	1	

Tabla 5.14: Sexta iteración.

Matriz Tabú				
0	0	0	0	3
0	0	0	3	4
0	7	4	0	5
7	5	0	0	6
6	0	0	0	0

Matriz de Frecuencias				
0	0	0	1	0
0	0	1	0	1
1	1	0	0	1
1	1	0	0	1
0	0	0	0	1

Tabla 5.15: Lista tabú e historial de la sexta iteración.

En la sexta iteración, el movimiento que proporciona el calendario $\pi = (0, 4, 3, 2, 1, 5, 6)$ es tabú, pero satisface el criterio de aspiración, por lo que se toma como punto inicial para la siguiente iteración. De manera análoga se sigue el procedimiento hasta la iteración 10 la cual tiene las tablas 5.16 y 5.17.

Si en alguna iteración ya no existen puntos admisibles, se tendría que utilizar ahora las funciones de memoria de plazo intermedio (intensificación)

Iteración	π	$F(\pi)$	Tabú = 1	Mejor Admisible
9	(0,2,1,4,3,5,6)	13500		***
10	(0,1,2,4,3,5,6)	14100	1	
	(0,2,4,1,3,5,6)	15500	1	
	(0,2,1,3,4,5,6)	14000	1	
	(0,2,1,4,5,3,6)	14700		*

Tabla 5.16: Décima iteración.

Matriz Tabú				
7	0	0	4	0
4	7	3	5	0
3	6	4	0	0
2	5	0	0	1
1	0	0	0	0

Matriz de Frecuencias				
1	1	0	1	0
2	1	1	1	1
1	2	1	1	1
1	1	1	0	1
0	0	0	0	0

Tabla 5.17: Lista tabú e historial de la décima iteración.

y de plazo largo (diversificación).

El contador de frecuencia muestra la distribución de los movimientos a través de las iteraciones. Se utiliza ese contador para diversificar la búsqueda, maniobrando dentro de nuevas regiones. Esta influencia de diversificación se restringe para operarse sólo en ocaciones particulares. En este caso, donde ningún movimiento de mejora admisible existe. En suma, las frecuencias definidas sobre diferentes subconjuntos de soluciones anteriores, particularmente subconjuntos de soluciones élites consistentes de óptimos locales de alta calidad encontrados en varios puntos, en el proceso de solución, proporcionan las estrategias complementarias de intensificación.

Una aproximación que está cercanamente unida a los orígenes de la BT y que proporciona un interjuego efectivo entre la intensificación y la diversificación es la *estrategia de oscilación*. La estrategia de oscilación consiste en moverse hasta llegar a una frontera. La vecindad se extiende, o bien, se permite que la selección de movimientos sea modificada a fin de cruzar la frontera y posteriormente se regresa en dirección opuesta. Esta acción crea una forma de oscilación que es la que le proporciona el nombre a dicho procedimiento. En muchas ocasiones, este tipo de movimiento oscilatorio es importante dependiendo tanto de la dirección de búsqueda como de reglas de movimientos modificadas.

La estrategia de oscilación opera mediante el moverse hasta pegarle a una frontera, representada por la factibilidad o un estado de construcción que normalmente puede representarse por un punto donde el método puede parar. En vez de parar, ahora bien, la definición de vecindad se extiende o el criterio de evaluación, para seleccionar movimientos, se modifica, para permitir que se pueda cruzar esa frontera. La aproximación procede para una profundidad específica más allá de la frontera y entonces se regresa. En este

punto la frontera de nuevo se aproxima y se cruza, esta vez desde la dirección opuesta, procediendo a un nuevo punto en turno. El proceso de aproximar y cruzar repetidamente la frontera desde diferentes direcciones crea una forma de oscilación que es la que le da el nombre a la estrategia. El control sobre esta oscilación se establece mediante la generación de evaluaciones y reglas de movimientos modificadas, dependiendo de la región en la cual se está actualmente navegando y de la dirección de la búsqueda.

Un ejemplo simple de esta aproximación ocurre para el problema de la mochila multidimensional donde los valores de las variables cero-uno se cambian de 0 a 1 hasta alcanzar la frontera de factibilidad. El método continua dentro de la región infactible utilizando el mismo tipo de cambios, pero con un evaluador modificado. Después de un número seleccionado de pasos, la dirección se invierte mediante cambiar las variables de 1 a 0. El criterio de evaluación se maneja para mejorar y varia de acuerdo a cuando el movimiento es de más a menos infactible o de menos a más infactible y se acompañan, mediante restricciones asociadas, sobre cambios admisibles para los valores de las variables.

Ahora bien, para incorporar la estrategia de oscilación, no necesariamente se tiene que definir en términos de factibiliad, sino que puede definirse donde la búsqueda parece gravitar. La oscilación consiste en forzar la búsqueda a movimientos fuera de esta región y el permitir regresarse a la región, ofreciendo de esta manera una forma efectiva para eliminar entrampamientos suboptimales en las búsquedas estándares.

5.3.2 Memoria de Plazo Intermedio y Largo

Como se ha visto, el método de la BT empieza con una solución factible inicial y en el proceso de ejecución, el procedimiento actualiza a los arreglos y elementos de la función memoria. Entonces el proceso se repite hasta que el criterio de terminación se encuentra.

En el método de BT descrito en el ejemplo, el *mejor* movimiento que se realiza en cada iteración se especifica como el movimiento admisible con el menor valor objetivo. Ahora bien, esta estrategia no garantiza que el movimiento seleccionado permita la búsqueda en la dirección de la solución optimal, por lo que se requiere de técnicas que nos permitan integrar las estrategias de intensificación y diversificación de manera efectiva, basándose sobre las funciones de memoria de plazo intermedio y largo de la BT. En otras

palabras, es de importancia vital el *mirar* la *dependencia regional* de buenos criterios de decisión, no sólo en términos de movimientos de mejoramiento y no mejoramiento.

5.3.3 Intensificación y Diversificación Regional.

La fase de intensificación proporciona una forma simple para enfocar la búsqueda alrededor de la mejor solución (o conjunto de soluciones élites) hasta el momento.

Para entender la importancia de estos recursos de la BT, consideremos dos ejecuciones del ejemplo numérico de calendarización considerado anteriormente, pero variando los valores de algunos de los parámetros iniciales.

Tomemos los siguientes datos inciales, los que dan paso a lo que llamaremos ejecución 1:

- Punto Inicial $\pi_0 = (0, 5, 4, 3, 2, 1, 6)$.
- $F(\pi_0) = 26600$.
- Longitud_tabú = 7.
- Distancia máxima = 2.

Como se puede observar en la tabla 5.18, en la iteración 2 se llegó a un óptimo local, el cual está dado por el calendario $\pi = (0, 3, 2, 1, 4, 5, 6)$ con un valor objetivo de 14900, en la iteración 3 también se llega a otro óptimo local dado por el calendario $\pi = (0, 1, 2, 3, 4, 5, 6)$ con igual valor objetivo, pero en este último caso se tiene que el punto es tabú y, además, no se tienen soluciones admisibles. Se puede entonces pensar en escoger a este último óptimo local para intensificar la búsqueda dentro de la región por lo que se toma como punto de arranque y se inicializa la tabla tabú para continuar la búsqueda; siguiendo el proceso, se puede observar que en la iteración 10 se cae en un ciclo y no existe mejoramiento, por lo que se puede considerar que se está en un valle profundo.

Ahora bien, en este ejemplo, la búsqueda pudo estar muy restringida debido al valor de la longitud tabú que es muy alto, por lo que se realizó una segunda corrida, que llamaremos ejecución 2 y que tiene los siguientes parámetros iniciales:

Iteración	π	$F(\pi)$	Tabú = 1	Mejor Admisible
0	(0,5,4,3,2,1,6)	26600		*
1	(0,5,2,3,4,1,6)	25500		*
	(0,3,4,5,2,1,6)	26100		
	(0,5,4,1,2,3,6)	26600		
2	(0,3,2,5,4,1,6)	20700	1	
	(0,3,2,1,4,5,6)	14900		*
	(0,3,4,5,2,1,6)	26100	1	
3	(0,1,2,3,4,5,6)	14900	1	
	(0,3,4,1,2,5,6)	19600	1	
	(0,3,2,5,4,1,6)	20700	1	

Tabla 5.18: Iteraciones de la ejecución 1.

- Punto Inicial $\pi_0 = (0, 5, 4, 3, 2, 1, 6)$.
- $F(\pi_0) = 26600$.
- Longitud_tabú = 3.
- Distancia máxima = 2.

Se observa en la tabla 5.19 que en la iteración 3 se alcanza un óptimo local y en la iteración 4 se alcanza el otro óptimo local que constituyen un agujero negro, pero en esta corrida a diferencia de la anterior, se tiene un punto admisible el cual se toma para proseguir con la búsqueda, pero siguiendo el proceso se tiene que en las iteraciónes 13 y 14 se vuelven a alcanzar los óptimos locales y se inicia un ciclo a partir de la iteración 20.

Las dos ejecuciones anteriores indican la necesidad de contar con elementos que nos permitan salir de este tipo de entrampamientos suboptimales, por lo que se debe de recurrir a un análisis retrospectivo del proceso de búsqueda que nos pueda proporcionar un reconocimiento de patrones de comportamiento para poder identificar regiones no visitadas.

A manera de ejemplo se considera la matriz de frecuencias de la ejecución 2 después de 20 iteraciones, donde las filas indican las posiciones y las columnas los trabajos (ver tabla 5.20).

Iteración	π	$F(\pi)$	Taú = 1	Mejor Admisible
0	(0,5,4,3,2,1,6)	26600		*
1	(0,5,2,3,4,1,6)	25500		*
	(0,3,4,5,2,1,6)	26100		
	(0,5,4,1,2,3,6)	26600		
2	(0,3,2,5,4,1,6)	20700	1	*
	(0,5,4,3,2,1,6)	26600		
	(0,5,2,1,4,3,6)	22600		
3	(0,5,2,3,4,1,6)	25500	1	
	(0,3,4,5,2,1,6)	26100	1	
	(0,3,2,1,4,5,6)	14900		*
4	(0,1,2,3,4,5,6)	14900	1	
	(0,3,4,1,2,5,6)	19600		*
	(0,3,2,5,4,1,6)	20700	1	

Tabla 5.19: Iteraciones de la ejecución 2.

Posiciones	Trabajos				
1	2	0	3	0	2
2	0	4	0	3	0
3	4	0	4	0	5
4	0	3	0	4	0
5	2	0	2	0	2

Tabla 5.20: Matriz de frecuencias de la ejecución 2.

Iteración	π	$F(\pi)$	Tabú = 1	Mejor Admisible
0	(0,2,1,5,3,4,6)	16300		*
1	(0,2,1,4,3,5,6) (0,5,1,2,3,4,6) (0,2,3,5,1,4,6)	13500 23400 18500		***

Tabla 5.21: Iteraciones del ejemplo 1.

Una forma de *aprender* qué regiones no se han visitado, es observar la frecuencia con la cual un cierto trabajo no se ha seleccionado para una posición particular, así por ejemplo, en la 5.20 se tiene que el trabajo 1 no se ha localizado en las posiciones 2 y 4, que el trabajo 3 se ha localizado más frecuentemente en la posición 3 de los calendarios, etcétera.

A partir de la matriz histórica de la búsqueda se pueden generar calendarios que se utilizan como puntos de arranque para nuevos procesos de búsqueda. A manera de ilustración se tienen de manera empírica los ejemplos dados a continuación, llamados ejemplo 1, 2 y 3.

Ejemplo 1.

- Punto Inicial: $\pi_0 = (0, 2, 1, 5, 3, 4, 6)$.
- $F(\pi_0) = 16300$.
- Longitud_tabu = 3.
- Distancia máxima = 2.

Ejemplo 2.

- Punto Inicial $\pi_0 = (0, 2, 3, 5, 1, 4, 6)$.
- $F(\pi_0) = 18500$.
- Longitud_tabu = 3.
- Distancia máxima = 2.

Iteración	π	$F(\pi)$	Tabú = 1	Mejor Admisible
0	(0,2,3,5,1,4,6)	18500		*
1	(0,2,1,5,3,4,6)	16300		*
	(0,5,3,2,1,4,6)	23100		
	(0,2,1,5,3,4,6)	17300		
2	(0,5,1,2,3,4,6)	23400		
	(0,2,3,5,1,4,6)	18500	1	
	(0,2,1,4,3,5,6)	13500		***

Tabla 5.22: Iteraciones del ejemplo 2.

Ejemplo 3.

- Punto Inicial $\pi_0 = (0, 5, 1, 4, 3, 2, 6)$.
- $F(\pi_0) = 24500$.
- Longitud_tabu = 3.
- Distancia máxima = 2.

Ahora bien, la diversificación se restringe para operarse sólo en ocasiones particulares. En este último ejemplo, se seleccionan las ocasiones donde ningún movimiento de mejora admisible existe. Por lo general, se utiliza la información de la frecuencia para penalizar a los movimientos que no mejoran la búsqueda mediante el asignar una penalización grande a intercambios de pares con mayores contadores de frecuencia.

Una implantación sencilla de tal técnica se puede realizar de la siguiente manera: primero, se cuenta el número de veces que cada movimiento, digamos m, se ha realizado con tal de calcular su frecuencia $f(m)$. Entonces una penalización $p(m)$ se asocia a cada movimiento de la siguiente manera:

$$p(m) = \begin{cases} 0 & \text{si } m \text{ alcanza un criterio de aspiración} \\ wf(m) & \text{de otra forma.} \end{cases}$$

donde w es una constante. El peso w depende del problema, del tipo de movimiento y de la vecindad. Glover y Laguna [49] indican que en muchas aplicaciones se ha observado que este peso es aproximadamente proporcional

Iteración	π	$F(\pi)$	Tabú = 1	Mejor Admisible
0	(0,5,1,4,3,2,6)	24500		*
1	(0,4,1,5,3,2,6)	23300		*
	(0,5,3,4,1,2,6)	27800		
	(0,5,2,1,3,4,6)	23400		
2	(0,5,1,4,3,2,6)	24500	1	
	(0,4,3,5,1,2,6)	26200		
	(0,4,1,2,3,5,6)	18900		*
3	(0,2,1,4,3,5,6)	13500	1	***
	(0.4.3.2,1,5,6)	19800		
	(0,4,1,5,3,2,6)	23300	1	

Tabla 5.23: Iteraciones del ejemplo 3.

a la raíz cuadrada del tamaño de la vecindad multiplicada por la desviación estándar del valor (sin penalización) de cada movimiento ejecutado durante la búsqueda.

Como se puede observar, la función de penalización depende directamente del criterio de aspiración, por lo que ahora nos enfocaremos a este concepto.

5.3.4 Criterios de Aspiración

Los criterios de aspiración se introducen en la BT para determinar cuándo las restricciones tabú pueden sobrellevarse para remover una clasificación tabú que, de otra manera, se aplicaría a un movimiento. El uso apropiado de tales criterios puede ser muy importante para que un método BT alcance sus mejores niveles de realización.

En las primeras aplicaciones de la BT se aplicó tan solo un tipo sencillo de criterio de aspiración, consistente de remover una clasificación tabú a un movimiento si éste permitía una solución mejor que la encontrada hasta el momento, tal regla se ilustró en los ejemplos de las secciones anteriores.

En [49], se indica que las aspiraciones son de dos clases: *aspiraciones de movimiento* y *aspiraciones de atributo*. Una aspiración de movimiento, cuando se satisface, revoca la clasificación tabú del movimiento. Una aspiración de atributo, cuando se satisface revoca el estado tabú del atributo.

En este último caso el movimiento puede o no cambiar su clasificación tabú, dependiendo de si la restricción tabú puede activarse por más de un atributo. Así entonces se pueden tener los siguientes criterios de aspiración:

- *Aspiración por Defecto*: Si todos los movimientos posibles son clasificados como tabú, entonces el movimiento *menos tabú* se selecciona.

- *Aspiración por Objetivo*: Una aspiración de movimiento se satisface, permitiendo que un movimiento m sea un candidato para seleccionarse si $F(m) <$ mejor costo.

- *Aspiración por Dirección de Búsqueda*: Un atributo de aspiración para un estado e se satisface si la dirección en e proporciona un mejoramiento y el actual movimiento es un movimiento de mejora.

5.4 Ejemplo: Recolección de Desechos Sólidos

La región marina (Figura 5.9) representa en la actualidad una zona de gran importancia para la economía del país, ya que de ella se extrae el más alto porcentaje de la producción nacional de hidrocarburos.

La actividad en las plataformas marinas se realiza durante las 24 horas del día de los 365 días del año, generándose una gran cantidad de desechos industriales y domésticos, clasificándose éstos en biodegradables (restos de alimentos) y no biodegradables (chatarra, plásticos, madera, etc.).

Para el tratamiento de estos desechos, se cuenta con equipos anticontaminantes abordo de las plataformas como son: trituradores de alimentos, compactadores e incineradores de basura.

Solamente los restos de alimentos son vertidos al mar después de ser triturados y el volumen de desechos restante se enviaba a la terminal marítima de Dos Bocas, Tabasco, contando para ello con el apoyo de barcos abastecedores de gran capacidad.

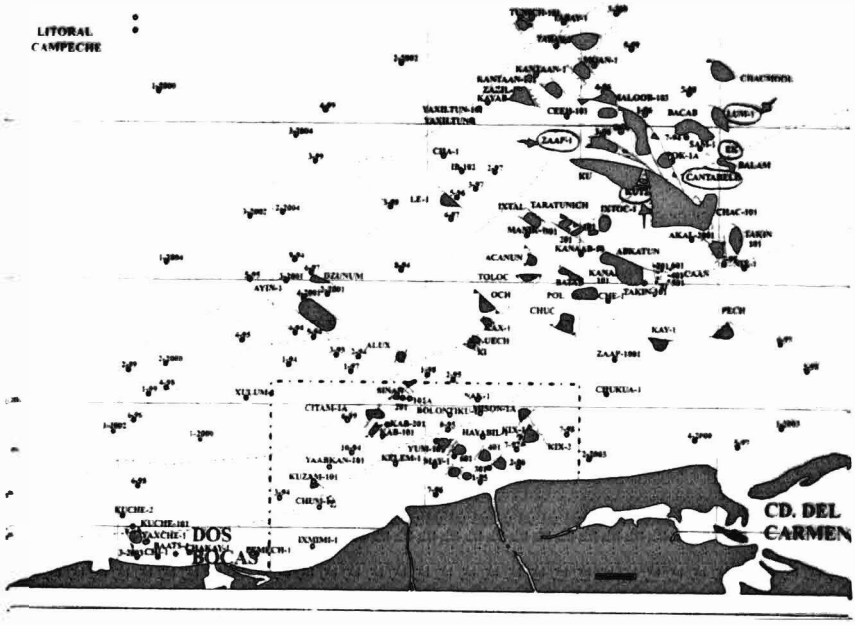


Figura 5.9: Región Marina de la zonda de Campeche.

5.4.1 Objetivo

De acuerdo con la política de Petróleos Mexicanos(PEMEX), para optimizar el aprovechamiento de sus recursos, se plantea la elaboración de un estudio en función del costo de transportación marina de todo el material que se genera en las plataformas localizadas en la sonda de Campeche, por lo que se deseaba conocer cuales eran las rutas óptimas de recolección de desechos y los resultados nos permitirían considerar la posibilidad de construir un centro de acopio de desechos sólidos en Cd. del Carmen, Campeche.

5.4.2 Información

Muchos datos son ficticios pero se ha tratado de que sean congruentes entre sí para que proporcionen una idea de la magnitud del problema real, el resto de la información ha sido obtenida de las diversas publicaciones de PEMEX.

De acuerdo con las actividades que se realizan, los complejos de producción generan en ese entonces en promedio 440 toneladas de desechos

	1	2	3	4	5	6	7	8	9	10	11
1	0	89.04	78.78	78.37	81.07	79.59	79.57	84.33	87.90	103.2	166.9
2	89.04	0	51.50	56.35	59.00	61.50	80.40	78.80	78.25	78.15	150.3
3	78.78	51.50	0	5.60	8.50	10.80	29.90	28.23	28.00	31.90	104.7
4	78.37	56.35	5.60	0	3.80	5.40	24.60	22.80	23.00	28.90	101.4
5	81.07	59.00	8.50	3.80	0	2.50	21.80	19.60	19.10	25.11	98.30
6	79.59	61.50	10.80	5.40	2.50	0	19.60	17.70	17.60	25.30	96.70
7	79.57	80.40	29.90	24.60	21.80	19.60	0	4.60	9.40	25.80	86.50
8	84.33	78.80	28.23	22.80	19.60	17.70	4.60	0	4.80	21.10	83.80
9	87.90	78.25	28.00	23.00	19.10	17.60	9.40	4.80	0	16.10	81.10
10	103.2	78.15	31.90	28.90	25.11	25.30	25.80	21.10	16.10	0	73.40
11	166.9	150.3	104.7	101.4	98.30	96.70	86.50	83.80	81.10	73.40	0

Tabla 5.24: Distancias de las plataformas a Cd. del Carmen (Kms.).

sólidos por mes, que representaban el 70% del total de desechos generados en el área marina. La recolección de chatarra y desechos sólidos en cada complejo de producción se llevaba a cabo dos veces al mes, por un barco dedicado exclusivamente a esa actividad. Para cumplir con las normas internacionales, la capacidad de recolección debería ser 1.1 veces el volumen de desechos generados, por lo que los barcos recolectores deberían contar con una capacidad de 250 toneladas.

Las distancias entre las plataformas y los centros de acopio[2] se muestran en las Tablas 5.24 y 5.25.

5.4.3 Solución y Resultados del Ejemplo

Lo que se desea es conocer las rutas óptimas (trayectorias de costo mínimo, véase Figura 5.10), por lo que se plantearon dos problemas del agente viajero en la forma (P1) descrita anteriormente y se resolvieron utilizando la técnica de la búsqueda tabú, mediante un algoritmo básico con longitud tabú igual a 5 y una estructura de vecindades formada por las permutaciones hacia adelante de al menos una posición, se utilizó una PC386 sin coprocesador así como un paquete comercial donde se corrieron los métodos de ramificación y

[2]la notación es la siguiente: 1: Cd. del Carmen, 2: Rebombeo, 3: POL-A, 4:ABKATUN-N, 5: ABKATUN-D ,6: ABKATUN-A, 7: NOHOCH-A, 8: AKAL-C, 9: AKAL-J, 10: KU-A, 11: Cayo Arcas.

	1	2	3	4	5	6	7	8	9	10	11
1	0	70.60	120.0	125.2	130.0	133.1	149.0	148.0	148.1	148.7	227.1
2	70.60	0	51.50	56.35	59.00	61.50	80.40	78.80	78.25	78.15	150.3
3	120.0	51.50	0	5.60	8.50	10.80	29.90	28.23	28.00	31.90	104.7
4	125.2	56.35	5.60	0	3.80	5.40	24.60	22.80	23.00	28.90	101.4
5	130.0	59.00	8.50	3.80	0	2.50	21.80	19.60	19.10	25.11	98.30
6	133.1	61.50	10.80	5.40	2.50	0	19.60	17.70	17.60	25.30	96.70
7	149.0	80.40	29.90	24.60	21.80	19.60	0	4.60	9.40	25.80	86.50
8	148.0	78.80	28.23	22.80	19.60	17.70	4.60	0	4.80	21.10	83.80
9	148.1	78.25	28.00	23.00	19.10	17.60	9.40	4.80	0	16.10	81.10
10	148.7	78.15	31.90	28.90	25.11	25.30	25.80	21.10	16.10	0	73.40
11	96.70	150.3	104.7	101.4	98.30	96.70	86.50	83.80	81.10	73.40	0

Tabla 5.25: Distancias de las plataformas a Dos Bocas (Kms.).

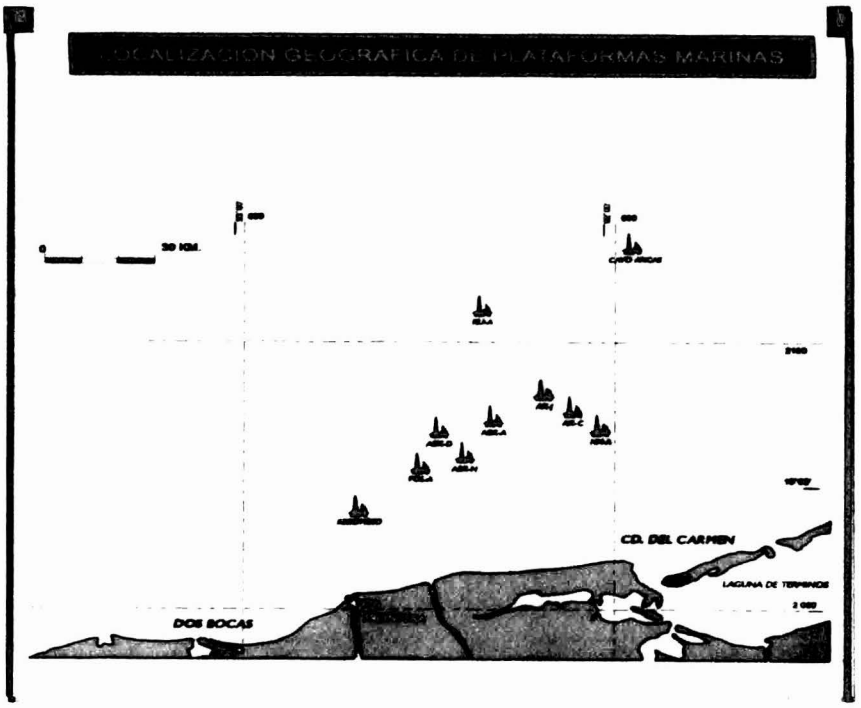


Figura 5.10: Región Marina de la zonda de Campeche puntos considerados en el ejemplo.

acotamiento (branch and bound [B.B]) y de la ruta del ángulo mayor (greatest

angle tour [G.A.T.]), encontrándose los resultados dados en la Tabla 5.26.

Cabe observar que el número de iteraciones utilizadas por la B.T. para encontrar los valores óptimos globales fueron de 4 para el problema de Cd. del Carmen y de 9 para el problema de Dos Bocas; además, para el problema de Cd. del Carmen, el método de la ruta del ángulo mayor fue del 97.7% de eficiencia respecto a la solución óptima global.

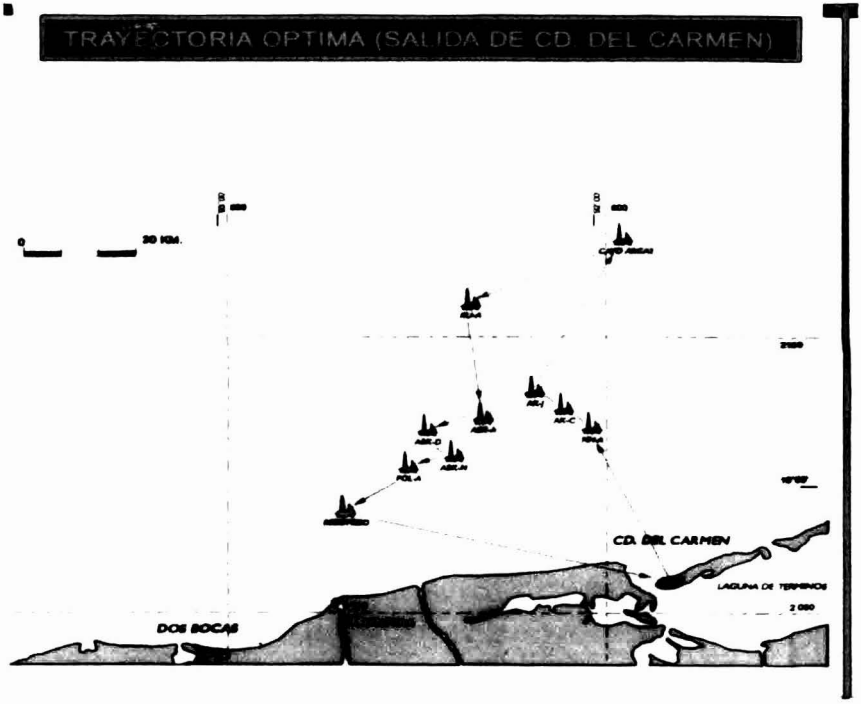


Figura 5.11: Trayectorias óptimas (de costo mínimo).

5.4.4 Conclusiones del Ejemplo

Como se puede observar, si se saliera de Cd. del Carmen se tendría un ahorro promedio por recolección de $54279.75, pero esta diferencia no justificaría la construcción de un centro de acopio en Cd. del Carmen en el corto plazo puesto que, este tipo de obra requiere cumplir con condiciones específicas establecidas por la normatividad nacional e internacional en materia ambiental, por tanto, se deberá seguir utilizando la terminal marítima de Dos Bocas como centro de acopio.

	Recorrido para recolección desde:	
	Cd. del Carmen	Dos Bocas
Distancia Total(Kms.)	421.21	464.15
Costo Total ($) a	60269.23	62530.88
B.T. Segundos	0.55	1.21
B.B. Segundos	3.0	33.61
G.A.T. Segundos	1.47	4.78

Tabla 5.26: B.T: Búsqueda Tabú, B.B: Branch and Bound, G.A.T: Trayectoria del ángulo mayor.

5.5 Ejemplo: Problema de Asignación Cuadrática

En su forma más simple, el problema de asignación cuadrática (QAP) puede describirse como sigue. Encontrar la asignación óptima de n objetos a n localidades para minimizar el producto acumulado de flujo entre cualesquiera dos objetos por las distancias entre cualesquiera dos localidades. De manera más precisa, si f_{ip} es el flujo entre los objetos i y p, y d_{jq} es la distancia entre las localidades j y q, el problema puede escribirse como:

$$\begin{array}{ll} \text{Minimizar} & \sum_i \sum_p \sum_j \sum_q f_{ip} d_{jq} x_{ij} x_{pq}, \\ \text{s.a.} & \sum_j x_{ij} = 1 \quad \forall i, \\ & \sum_i x_{ij} = 1 \quad \forall j, \\ & x_{ij} \in \{0,1\}. \end{array}$$

La variable x_{ij} es diferente de cero si el objetivo i es asignado a la localidad j y es cero de otra forma. La función a optimizar es cuadrática y no convexa, por ejemplo, existe cierto número de óptimos locales; además, el conjunto factible es un conjunto de permutaciones.

Una forma sencilla de interpretar el problema planteado es suponer que existen n sitios disponibles y se van a construir n edificios en estos lugares, entonces si f_{ip} es el número de gentes por unidad de tiempo que viajarán entre los edificios i y p, y d_{jq} es la distancia entre los sitios j y q, lo que se desea es encontrar la asignación de los sitios de construcción de manera que la distancia total recorrida se minimice.

Un aspecto interesante de este problema es el hecho de que cuando una de las matrices de la función objetivo se restringe a ser una matriz de permutaciones cíclica, el problema de asignación cuadrático se reduce al problema del agente viajero.

Una formulación equivalente del problema es:

$$F(\pi) = \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij} d_{\pi(i)\pi(j)} \rightarrow \min_{\pi} \tag{5.1}$$

donde π es una permutación del conjunto $N = 1, 2, \ldots, n$. Equivalentemente, se desea encontrar una permutación π del conjunto N que minimice el valor de F.

5.5.1 Elementos de la BT para el QAP

Las siguientes definiciones describen los elementos de la búsqueda tabú para el QAP. Un *movimiento* es una transición de una permutación a otra. Un atributo de un movimiento, en este caso, es simplemente un par no ordenado (i, j) de objetos que cambian sus localidades. El valor de un movimiento es la diferencia entre los valores de la función objetivo después y antes del movimiento. Si éste valor es negativo, el movimiento proporciona una mejora. La función *mejor_movimiento* es una que identifica al par (i_{mejor}, j_{mejor}) para el cual el valor de movimiento es el más pequeño. El dominio de la función *mejor_movimiento* es el conjunto de movimientos admisibles. El mejor movimiento no tiene que ser necesariamente uno que mejore.

Una lista tabú de pares (i, j) de longitud *longitud_tabú* se construye y actualiza. Si un par (i, j) pertenece a la lista tabú para una iteración dada, no se permite el intercambio de los objetos i y j en esa iteración, a menos que se satisfaga el criterio de aspiración que en este caso se cumple si el intercambio de los objetos i y j proporciona un valor de la función objetivo estrictamente mejor que cualquiera de los obtenidos hasta el momento.

5.5.2 Estrategia Básica

La estrategia básica que se utilizará está dividida en cuatro fases: fase de inicialización, fase de búsqueda preliminar, fase de intensificación y fase de diversificación.

Cada iteración de la fase de búsqueda preliminar evalúa todos los posibles movimientos y determina el mejor movimiento valuado admisible. Si ninguna solución mejora después de cierto número prefijado de iteraciones, por ejemplo, $k_1 n$, donde k_1 es número natural, la fase de intensificación inicia, con una lista tabú vacía, desde la mejor solución encontrada en la región actual. Si después de $k_2 n$ iteraciones con k_2 número natural, no se ha encontrado una mejor solución, la fase de diversificación inicia. El algoritmo termina después de que la búsqueda se ha diversificado a un número dado k_3 de regiones.

Para la diversificación se utiliza la frecuencia basada en la memoria de largo plazo. Esta memoria de largo plazo puede representarse por medio de una matriz $n \times n$, donde cada elemento LT_{ij} almacena el número de veces en que cada entrada de una permutación toma el valor de 1 en todas las soluciones examinadas por la búsqueda; de esta manera, si un cierto porcentaje de las entradas con mayor frecuencia se guardan como tabú para un número suficiente de iteraciones, la búsqueda se forzará para la exploración de nuevas regiones. Alternativamente, una nueva solución inicial puede construirse basada en la matriz de largo plazo.

Estrategia Básica

Fase 1 *Inicialización:* Genere solución inicial y continúe.

Fase 2 *Búsqueda preliminar:*

1. Evalúe todos los movimientos, determine y realice el mejor.
2. Actualice.
3. Si no hay mejoramiento en $k_1 n$ iteraciones vaya a Fase 3.

Fase 3 *Intensificación:*

1. Vaya a la mejor solución encontrada e inicialice la lista tabú.
2. Evalúe todos los movimientos, determine y realice el mejor.
   (a) Actualice.
   (b) Si no hay mejoramiento en $k_2 n$ iteraciones vaya a Fase 4.

Fase 4 *Diversificación:*

1. Determine las restricciones tabú de la memoria de plazo largo.

2. Evalúe todos los movimientos, determine y realice el mejor.
3. Actualice.
4. Si no hay mejoramiento en $k_3 n$ iteraciones:
   (a) Inicialice con la mejor solución.
   (b) Elimine las restricciones tabú adicionales debidas a la memoria de largo plazo, actualice el número de reinicios y vaya a la Fase 2.

Una vez que se han identificado los movimientos factibles y se ha calculado el correspondiente valor de la función objetivo, el estado tabú de los candidatos factibles se prueba. Laguna y Kelly [73] proponen algunos criterios de aspiración, entre los que se encuentran los que a continuación se exponen: suponga que en una iteración dada (*iter*) al valor de $\pi(s)$, donde $s = s_i$, cambia de i a i', entonces a la localidad $\pi(i)$, le está prohibido ocupar la posición i durante un número específico de iteraciones. La matriz *tiempo_tabú* se utiliza para forzar esta restricción tabú. El elemento $(i, \pi(i))$ de la matriz *tiempo_tabú* contiene el número de iteración en la cual se le permite de nuevo a la localidad $\pi(i)$ ser asignada a la posición i. Si el valor de la función objetivo de la solución anterior al movimiento es mejor que el valor de la función objetivo después de éste, entonces el tiempo tabú se calcula como:

$$\text{tiempo_tabú}(i, \pi(i)) = 0.25n^2,$$

de otra forma:

$$\text{tiempo_tabú}(i, \pi(i)) = 0.75n^2.$$

Este es un criterio de aspiración implícito, donde a los elementos que contribuyen a soluciones con *buenos* valores de la función objetivo se les permite regresar más rápidamente para la prueba de soluciones actuales. El máximo tiempo que un movimiento puede clasificarse como tabú es el equivalente al 75% del tamaño de la estructura de *tiempo_tabú*. Esto significa que, en una iteración dada, un mínimo del 25% de los movimientos posibles no se clasifican como tabú.

Ahora bien, un criterio de nivel de aspiración de tipo explícito es el siguiente: el estado tabú de un movimiento se elimina si el movimiento permite la búsqueda a una nueva solución *mejor de la región*. La mejor solución de la región es aquella con el mejor valor de la función objetivo durante digamos las

n^2 últimas iteraciones. Observe que este nivel de aspiración es más flexible que el utilizado típicamente en las implantaciones de BT, donde la mejor solución sobre todas sirve para propósitos similares.

Otro criterio explícito es una aspiración por *dirección del movimiento.* Siguiendo a Laguna y Kelly [73], para implantar este criterio es necesario construir una matriz *dirección_tabú.* Si el movimiento $(i, \pi(i))$ llega a ser tabú durante una fase de mejoramiento, entonces el elemento $(i, \pi(i))$ de la matriz dirección_tabú contiene el número de la iteración de cuando la fase de mejoramiento comenzó, de otra forma, al elemento se le da el valor de cero. El movimiento candidato factible $(i, \pi(i))$ es elegible para seleccionarse en la iteración *iter* si F(movimiento posterior) $\neq F$(movimiento anterior) y ocurre al menos una de las siguientes condiciones:

- tiempo_tabú$(i, \pi(i)) <$ iter.

- F(movimiento posterior) $< F$(mejor de la región).

- dirección_tabú$(i, \pi(i)) \neq 0$ y dirección_tabú$(i, \pi(i))$ = Fase actual,

donde fase_actual es cero si la búsqueda se encuentra en una fase de no mejoramiento o el número de la iteración donde la actual fase de mejoramiento comenzó. El mejor movimiento en cualquier iteración es el movimiento elegible con mejor valor de movimiento.

5.5.3 Manejo Dinámico de la Lista Tabú

El *manejo de la lista tabú* (LT) significa la actualización de ésta, por ejemplo, la decisión de cuántos y cuáles movimientos se pondrán como tabú para una iteración de la búsqueda. La mayoría de las implantaciones de la búsqueda tabú utilizan un manejo de la lista tabú de tipo estático. De manera más precisa, los movimientos permanecen como tabú durante un número de iteraciones fijo, por lo que la eficiencia del algoritmo depende de la elección de la duración del estado tabú o, equivalentemente, de la longitud de la lista tabú.

El manejo de la lista tabú de tipo dinámico permite el examen más detallado de la región factible, por lo que es posible romper ciclos y diversificar la búsqueda.

Existen varios métodos de lista tabú dinámica, entre los que destacan: el método de la *secuencia de Cancelación* (CSM) y el método de *eliminación inversa* (REM) propuestos en [48].

Ahora bien, también se puede manejar una longitud de LT de manera dinámica como la propuesta dada en [24], donde la longitud de la lista tabú varía de manera dinámica a través de diferentes configuraciones pasando de una a otra si ningún mejoramiento se encuentra en un número dado de iteraciones. Este conjunto de configuraciones permite un examen más detallado de la región factible mediante el incremento y decremento del número de movimientos tabú actuales vía las configuraciones. Además se propone un componente aleatorio que actúa cada vez que la lista tabú cae en la configuración 1 como un elemento de seguridad adicional contra el ciclado.

5.5.4 Experiencia Computacional

Varias versiones de BT que se describen a continuación se probaron sobre el conjunto estándar de problemas de tamaño n = 5, 6, 7, 8, 12, 15, 20, 30 dados por Nugent *et. al.* (1967), los cuales se utilizan de manera amplia para probar la eficiencia de los algoritmos.

Para todo el conjunto de problemas se encontraron las mejores soluciones reportadas en la literatura.

Las versiones que se utilizaron de la búsqueda tabú tienen las siguientes características:

- *BT1:* Estructura básica de BT, por ejemplo, se consideró la longitud de la lista tabú como un parámetro constante, el criterio de nivel de aspiración fue el estándar de la literatura, es decir, el nivel de aspiración se satisface si el movimiento a realizarse mejora el valor de la función objetivo con respecto de todos los movimientos realizados.

- *BT2:* Se consideró una estructura básica de la BT y se agregó el criterio de nivel de aspiración proporcionado por *tiempo_tabú* dado en la sección 5.5.2.

- *BT3:* En este marco de la BT se consideró el criterio de nivel de aspiración estándar, así como, un manejo de la lista tabú de tipo dinámico, parecido al propuesto en para el caso de programación en paralelo y que se describe a continuación:

Manejo Dinámico Propuesto: Si no existe mejoramiento, entonces el movimiento se considera tabú si:

$$\liminf \leq \text{tabu}(i,j) \leq \limsup,$$

donde

$$\liminf = \begin{cases} 1 & \text{si } numiter \mod 8 < 5, \\ ltabu(numiter \mod 8)(0.2) & \text{de otra forma,} \end{cases}$$

$$\limsup = \begin{cases} ltabu((numiter \mod 8) - 4)(0.2) & \text{si } numiter \mod 8 < 5, \\ ltabu & \text{de otra forma,} \end{cases}$$

siendo *numiter* el número de iteración y *ltabu* la longitud tabú, ver [24].

Los resultados obtenidos se muestran en la tabla 5.27. Los valores entre paréntesis y el número de iteraciones respectivo son los mejores valores que se encontraron para cuando se excedía de un número fijo predeterminado de iteraciones *max_iter*, el cual dependía del tamaño de los problemas, siendo los siguientes:

n	5	6	7	8	12	15	20	30
max_iter	15	15	25	25	700	2500	7500	12000

Para mayores detalles véase [32].

5.6 Conclusiones

En este capítulo se enfocó la atención al análisis de la técnica de la BT, observando que esta técnica es aplicable a todo tipo de problemas de programación combinatoria.

Se revisó el problema de calendarización con costos de penalización por retraso y costos de actualización, presentándolo de manera original, como marco para introducir los diferentes elementos de la técnica de la BT.

Se presentó un ejemplo real de *recolección de basura sólida* en una de las principales compañías mexicanas, PEMEX, y se presentaron los resultados alcanzados respecto de otras metodologías.

n	Pto. Inic.	Valor Inic.	Long. Tabú	BT1 Valor	BT1 Iter.	BT2 Valor	BT2 Iter.	BT3 Valor	BT3 Iter.	Valor Mejor
5	1	66	5	(58)	1	50	11	50	14	50*
	2	58	5	(58)	0	50	11	50	11	50*
	3	72	5	50	1	50	1	50	1	50*
	4	68	5	(52)	6	50	12	50	4	50*
	5	82	5	(52)	5	50	12	50	12	50*
6	1	86	4	86	0	86	0	86	0	86*
	2	110	4	86	2	86	2	86	2	86*
	3	108	4	86	6	92	1	86	6	86*
	4	116	4	86	1	86	1	86	1	86*
	5	116	4	86	8	86	11	86	8	86*
7	1	174	5	148	23	148	21	148	23	148*
	2	240	5	148	10	148	10	148	10	148*
	3	216	5	148	4	148	4	148	4	148*
	4	228	5	148	18	148	18	148	18	148*
	5	202	5	148	19	148	19	148	19	148*
8	1	214	6	214	10	214	16	214	10	214*
	2	322	6	214	5	214	5	214	5	214*
	3	290	6	214	7	214	15	214	7	214*
	4	320	6	214	5	214	5	214	5	214*
	5	288	6	214	4	214	4	214	4	214*
12	1	784	9	(586)	57	578	683	578	368	578*
	2	784	9	578	377	(586)	110	578	18	578*
	3	874	9	578	12	578	16	578	11	578*
	4	850	9	(586)	23	578	84	578	53	578*
	5	746	9	578	5	578	5	578	4	578*
15	1	1448	10	1150	1495	1150	46	1150	200	1150*
	2	1612	10	1150	1468	1150	375	1150	231	1150*
	3	1596	10	1150	2368	(1152)	53	(1152)	51	1150*
	4	1610	10	1150	1124	(1152)	181	(1152)	305	1150*
	5	1626	10	(1152)	33	(1152)	32	(1152)	32	1150*
20	1	3444	12	2580	7435	2570	559	2570	1014	2570
	2	3302	12	2570	5626	2570	954	2570	1841	2570
	3	3540	12	2570	1869	2570	701	2570	2197	2570
	4	3456	12	2570	918	2570	890	2570	887	2570
	5	3322	12	2570	1191	2570	613	2570	2408	2570
30	1	8060	18	6124	11048	6124	3628	(6158)	171	6124
	2	7758	18	6124	7294	(6128)	6223	6124	10692	6124
	3	8172	18	(6128)	972	6124	9842	(6148)	7740	6124
	4	7648	18	(6128)	2713	6124	3995	(6128)	1049	6124
	5	8224	18	6124	5748	6124	3839	(6148)	7881	6124

Tabla 5.27: Resultados comparativos para el problema de asignación cuadrático.

Un aspecto de interés fue el de la identificación de valles profundos, concepto de suma importancia para la eliminación del ciclado y/o soluciones suboptimales.

Aunque nuestro trabajo se basa en un número limitado de estrategias de búsqueda de vecindades así también como de soluciones iniciales, es interesante observar como el uso de criterios de niveles de aspiración proporcionan métodos más robustos de la BT en cuanto a la eficiencia de soluciones para problemas clásicos considerados como difíciles.

Las adaptaciones del método de BT son muy variadas y aún con las estructuras más sencillas del mismo se pueden obtener *buenas soluciones.*

La técnica de BT proporciona excelentes resultados para problemas de tipo combinatorio y existen por explorar muchas otras adaptaciones para obtener cada vez mayor eficiencia.

5.7 Lineamientos de BT para Problemas Continuos

En esta sección se presentan los lineamientos generales para utilizar la Búsqueda Tabú cuando se tiene una función continua de la forma:

$$f(\vec{\theta}), \quad \text{sujeto a: } \vec{\theta} \in \Omega.$$

donde $\vec{\theta}$ es el vector de los parámetros de la función.

5.7.1 Detalles de Implementación

Como se observa en [110], primero se discretiza el espacio de parámetros mediante una malla de ancho h_k, siendo h_k más fina conforme se avance en las iteraciones. Así, un estado es un vector $\vec{\theta}$ del espacio de parámetros.

Un vecindario se genera por el desplazamiento en la malla hacia alguno de los estados contiguos, mediante la modificación por h_k en alguno de los parámetros del vector $\vec{\theta}$. Por ejemplo, en el caso donde un estado está dado por el vector tridimensional $\vec{\theta} = (\theta_1, \theta_2, \theta_3)$, entonces los vecinos son los seis estados:

$\vec{\theta}^1 = (\theta_1 + hk, \theta_2, \theta_3)$	$\vec{\theta}^2 = (\theta_1, \theta_2 + hk, \theta_3)$	$\vec{\theta}^3 = (\theta_1, \theta_2, \theta_3 + hk)$
$\vec{\theta}^4 = (\theta_1 - hk, \theta_2, \theta_3)$	$\vec{\theta}^5 = (\theta_1, \theta_2 - hk, \theta_3)$	$\vec{\theta}^6 = (\theta_1, \theta_2, \theta_3 - hk)$

Así, si el estado $\vec{\theta}$ es un vector p dimensional, entonces tiene $2p$ vecinos.

Se necesita estar revisando si el movimiento realizado cumple con las restricciones del problema, es decir, si $\vec{\theta} \in \Omega$, en caso contrario se necesita de algún mecanismo para que nos lleve a la región factible.

El movimiento consiste entonces en la elección tanto de uno de los parámetros, como de una dirección. De esta forma, si a la l-ésima coordenada de $\vec{\theta}$ se le añade h_k, lo cual podemos denotar como el movimiento $(l, +1)$, entonces codificaremos en la lista tabú el movimiento inverso $(l, -1)$ con el fin de impedir el regreso a la posición anterior de $\vec{\theta}$. Usaremos una lista tabú de tamaño p y emplearemos algún criterio de aspiración antes descrito.

Para una aplicación de la Búsqueda Tabú a problemas de funciones continuas y en particular para problemas de regresión no lineal se puede consultar [110].

5.8 Ejercicios

1. Suponga que la función objetivo es el número de unos en una cadena de longitud 4 de ceros y unos; se desea obtener el valor mínimo de esta función por medio de búsqueda tabú, defina para este problema:

   (a) La estructura de vecindades.

   (b) Explique cómo sería una función de memoria de corto plazo(lista tabú).

   (c) Proporcione un criterio diferente a cierto número de iteraciones, para que un movimiento permanezca como tabú.

   (d) Explique cómo implementaría una función de memoria de largo alcance.

   (e) Haga cuatro iteraciones del algoritmo de búsqueda tabú, tomando la solución inicial igual a (1,0,0,1).

2. Para cada uno de los ejercicios del i-ix, realice las siguientes actividades:

a Proponga un tipo de estructura que tendrían las soluciones factibles del problema.

b Proponga una estructura de vecindades respecto al inciso anterior.

c Explique cómo sería una función de memoria de corto plazo (lista tabú).

d Proporcione un criterio diferente a cierto número de iteraciones, para que un movimiento permanezca como tabú.

e Explique cómo implementaría una función de memoria de largo alcance.

f Proporcione un criterio de paro para la fase de intensificación regional diferente a un número máximo de iteraciones o de un tiempo máximo de cómputo.

g A partir de la función de memoria de largo plazo, ¿cómo encontraría puntos en otras regiones para realizar la intensificación regional?.

h Haga un reporte con todos sus análisis y resultados.

i El problema de conjunto independiente, visto en la sección 3.1.4.

ii El problema de coloración mínima, visto en la sección 3.1.5.

iii El problema de empaquetamiento, visto en la sección 3.1.6.

iv El problema de cubrimiento por vértices, visto en la sección 3.1.7.

v El problema del agente viajero visto en la sección 3.1.8.

vi El problema de clan máximo, visto en la sección 3.1.9.

vii El problema de corte máximo, visto en la sección 3.1.10.

viii El problema de satisfacibilidad, visto en la sección 3.1.11.

ix El problema de la mochila, visto en la sección 3.1.12.

3. Considere el problema de programación entera:

$$\begin{aligned} &\text{Max } z = 6x_1 + 8x_2 \\ &\text{Sujeto a}: 3x_1 + 4x_2 \leq 12 \\ &\qquad\qquad 5x_1 + 2x_2 \leq 10 \\ &\qquad\qquad x_1 \geq 0,\ x_2 \geq 0 \ \text{ y enteros} \end{aligned}$$

(a) Resuelva el problema gráficamente.

(b) Use búsqueda tabú para encontrar la solución óptima.

4. Un aserradero recibe troncos con una longitud de 20 pies cada uno y se cortan en longitudes más pequeñas para su venta a algunas compañías manufactureras. El aserradero tiene órdenes de compra para las siguientes longitudes: $l_1 = 3$ pies, $l_2 = 7$ pies, $l_3 = 11$ pies y $l_4 = 16$ pies. La compañía actualmente tiene un inventario de 2,000 troncos, cada uno, de 20 pies de longitud. Suponga que el aserradero actualmente tiene suficientes órdenes, de modo que su problema es determinar el patrón de corte que maximice sus beneficios. El beneficio que obtiene por cada uno de los cortes en longitud más pequeña es la siguiente:

Longitud (pies)	3	7	11	16
Beneficio ($)	1	3	5	8

cualquier patrón de corte es permisible con tal que la longitud no exceda 20 pies, es decir,

$$3d_1 + 7d_2 + 11d_3 + 16d_4 \leq 20$$

donde d_i es el número de piezas cortadas a la longitud $l_i, i = 1, 2, 3, 4$. Resuelva este problema usando el algoritmo de búsqueda tabú.

5. Una cadena de tiendas de deportes tiene tres tiendas en una ciudad. Un fabricante de pelotas de tenis le ofrece al director de la cadena de tiendas, 500 docenas de pelotas de tenis a un precio muy bajo. Las tres tiendas tienen una muy buena venta de artículos para tenis, por lo que el director decide comprar las pelotas. Así, el director ahora tiene el problema de determinar que cantidad de pelotas asignará a cada una de las tiendas. La siguiente tabla muestra el beneficio esperado por asignar 100, 200, 300, 400 o 500 docenas de pelotas a cada tienda

	Número de docenas de pelotas de tenis				
Tienda	100	200	300	400	500
1	$600	$1100	$1550	$1700	$1800
2	500	1200	1700	2000	2100
3	550	1100	1500	1850	1950

Suponga que los lotes no pueden ser separados en tamaños menores a 100 docenas.

Use búsqueda tabú para determinar la asignación óptima de las docenas de pelotas a las tres tiendas de deportes.

6. Una compañía contrata ocho nuevos empleados y debe determinar donde colocarlos. Existen cuatro actividades. La compañía ha preparado la siguiente tabla, que da el beneficio estimado para cada actividad en función del número de nuevos empleados que le sean asignados

	Número de nuevos empleados								
Actividades	0	1	2	3	4	5	6	7	8
1	22	30	37	44	49	54	58	60	61
2	30	40	48	55	59	62	64	66	67
3	46	52	56	59	62	65	67	68	69
4	5	22	36	48	52	55	58	60	61

(a) Use búsqueda tabú para determinar la asignación óptima de los nuevos empleados a las actividades.

(b) Suponga que únicamente seis nuevos empleados fueron contratados. ¿A qué actividades deben asignarse estos empleados?

7. Una empresa tiene a su disposición cinco proyectos para los próximos tres años. El valor presente de: los ingresos esperados, los gastos anuales y la disponibilidad de recursos disponibles durante estos años, de cada proyecto, se describen en la siguiente tabla:

Proyecto	Gastos anuales (millones)			Ingresos (millones)
	Año 1	Año 2	Año 3	
1	5	1	8	20
2	4	7	10	40
3	3	9	2	20
4	7	4	1	15
5	8	6	10	30
Recursos disponibles:	25	25	25	

El gerente desea seleccionar los proyectos que maximicen el valor presente neto de los ingresos de la empresa. Formule el modelo como

un modelo de optimización combinatoria y resuélvalo usando búsqueda tabú.

8. Use el algoritmo de búsqueda tabú para resolver el problema del agente viajero con la matriz de costos no simétrica que aparece abajo.

a $j =$	1	2	3	4	5	6	7
de $i = 1$	×	13	9	19	22	3	7
2	6	×	7	8	39	4	8
3	18	3	×	5	18	4	3
4	4	16	6	×	5	13	16
5	14	22	99	17	×	8	10
6	3	21	7	3	10	×	9
7	11	19	8	19	8	3	×

9. En una fábrica textil hay un telar especial que fabrica una gran variedad de tejidos de un tipo especial. El primer día del mes la fábrica tiene órdenes de 7 trabajos que pueden procesarse en éste, para $i = 1, ..., 7$, p_i, d_i, r_i son respectivamente el tiempo de proceso en días, día de entrega y el beneficio del trabajo i. Estos datos se dan en la siguiente tabla:

i	1	2	3	4	5	6	7
p_i	9	10	12	5	11	8	13
d_i	4	13	15	8	20	30	30
r_i	90	130	85	35	77	68	100

Si el trabajo i se acepta, éste se tiene que entregar en el día de entrega para ese trabajo (donde d_i es el número de días contados a partir del primer día del mes). Los trabajos son independientes y el telar puede procesar solamente un trabajo a la vez.

(a) Formular el problema de seleccionar los trabajos que serán aceptados maximizando el beneficio total sujeto a la restricción que todos los trabajos aceptados deben completarse en su respectiva fecha de entrega.

(b) Desarrolle un algoritmo de búsqueda tabú para obtener una buena solución a este problema.

10. Para cada uno de las siguientes funciones i-xiii, vistas en la sección 3.2, realice las siguientes actividades:

    a Proponga un tipo de estructura que tendrían las soluciones factibles del problema.

    b Proponga una estructura de vecindades respecto al inciso anterior.

    c Explique cómo sería una función de memoria de corto plazo(lista tabú).

    d Proporcione un criterio diferente a cierto número de iteraciones, para que un movimiento permanezca como tabú.

    e Explique cómo implementaría una función de memoria de largo alcance.

    f Proporcione un criterio de paro para la fase de intensificación regional diferente a un número máximo de iteraciones o de un tiempo máximo de cómputo.

    g A partir de la función de memoria de largo plazo, ¿cómo encontraría puntos en otras regiones para realizar la intensificación regional?.

    h Haga un reporte con todos sus análisis y resultados.

        i La función de Ackley
        ii La función RCOS de Branin
        iii La función de Easom
        iv La función de la esfera
        v La función de Goldstein-Price
        vi La función de Griewangk
        vii La función de Langermann
        viii La función de Michalewicz
        ix La función de Rastrigin
        x La función de Rosenbrock
        xi La función de Schaffer F6
        xii La función de Schwefel
        xiii Las función de las seis jorobas del camello

Capítulo 6

Inteligencia de Enjambre

La observación de la naturaleza ha permitido la propuesta de nuevos paradigmas computacionales. La gran variedad de problemas de optimización que ésta presenta, tales como: la supervivencia, la evolución de las especies, la búsqueda del alimento, entre otros, sugirieron diversos algoritmos computacionales que se inspiran en procesos naturales.

La inteligencia de enjambre es un tema que ha surgido en los últimos veinticinco años, debido al interés en resolver problemas de manera colectiva sin un control centralizado. Estos problemas ocurren, por ejemplo, en el ámbito de la robótica y los agentes. Se han visto fenómenos relevantes en la naturaleza como los siguientes:

1. Ciertas especies de hormigas rápidamente encuentran caminos más cortos desde su nido a un nuevo sitio de comida.

2. Parvadas de aves vuelan en formaciones complejas sin chocar entre sí.

3. Termitas construyen montículos de hasta nueve metros de altura con una temperatura regulada.

4. Abejas informan al resto del enjambre sobre lo promisorio y la ubicación de sitios de comida mediante un "baile".

De estos ejemplos, y muchos otros que se han observado, ha surgido la pregunta sobre la manera en la cual las especies comunican su conocimiento, y cómo logran tareas que desde afuera parecen bastante complejas. Uno de

los primeros trabajos fue una simulación de Craig Reynolds, llamado Boids, que mostró que se podría simular el movimiento de parvadas mediante tres reglas que operaban de manera local, pero que producía un resultado global complejo.

Muchos investigadores han sido inspirados en las soluciones ofrecidas por la naturaleza, y han desarrollado algoritmos como optimización por hormigas y optimización por enjambres de partículas, que estudiaremos en este capítulo, al igual que otros como optimización de bacterias, la búsqueda por difusión estocástica, y el algoritmo de colonias de abejas.

A este tipo de métodos se le conoce como *inteligencia de enjambre*, que corresponden a las técnicas heurísticas de vida artificial que están basadas en el estudio del comportamiento colectivo de poblaciones (enjambres) como las hormigas, las abejas, las termitas, etc., en dichas poblaciones, los miembros o agentes realizan una actividad de búsqueda de manera individual puesto que no existe un control centralizado que les indique cómo comportarse, pero a la vez existe una "comunicación" entre los agentes que permite que la población actúe como un todo para alcanzar un cierto objetivo, el cual es por lo general la obtención de alimento para todo el enjambre.

En este capítulo se presentan los lineamientos de las técnicas conocidas como optimización por enjambres de partículas y optimización por colonia de hormigas, se revisan los trabajos [4], [5], [17], [34], [38], [41], [42], [65], [66], [86] y [105] entre otros.

6.1 Optimización por Enjambre de Partículas

En esta sección se presenta el método de Optimización por Enjambre de Partículas (PSO, del inglés Particle Swarm Optimization). Se proporcionan las líneas generales de esta técnica y posteriormente se presentarán dos aplicaciones: la primera para problemas de inventarios multiproducto con costos de reorden combinados y, la segunda, para problemas de regresión no lineal. En ambos ejemplos se proporcionarán resultados que muestran su competitividad respecto de otros métodos.

Además, se presenta la bondad de la técnica de PSO como un procedimiento de mejora de búsqueda. La sección se desarrolla como sigue: se presenta una introducción donde se considera PSO como un procedimiento

para realizar una fase de mejoramiento y se realiza un análisis comparativo contra otros métodos; mostrando la experiencia computacional; posteriormente, se describe el modelo de inventarios multiproducto para el caso de demanda determinística, así como algunos otros algoritmos que resuelven el problema; se sigue con la descripción del modelo de regresión no lineal; finalmente, se proporcionan algunas conclusiones y líneas de investigación.

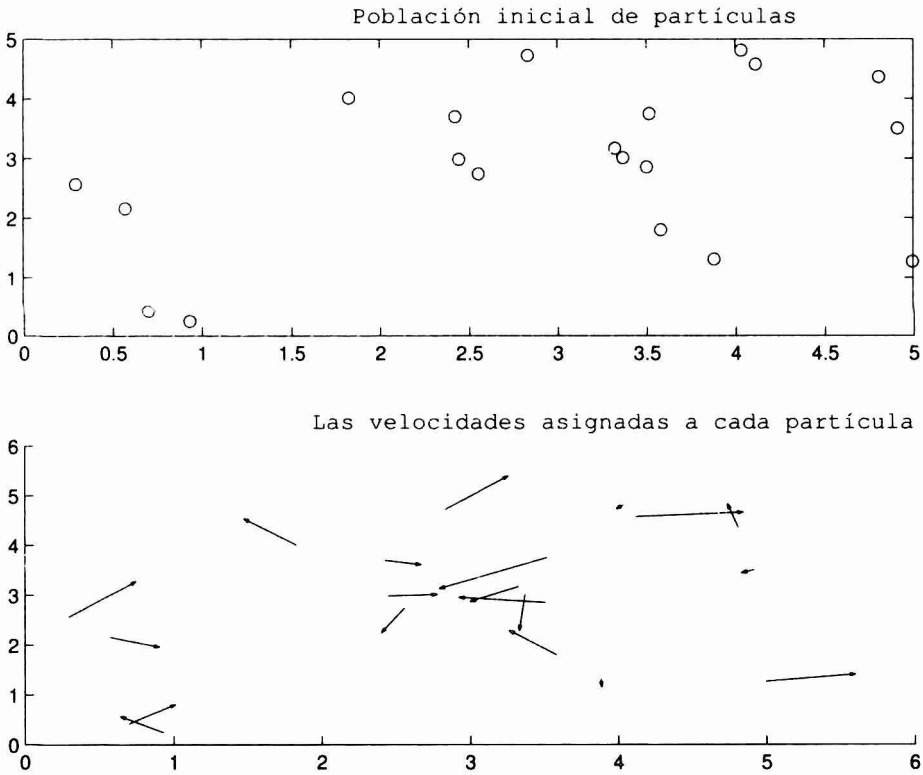


Figura 6.1: Enjambre (subconjunto de puntos en el conjunto factible).

6.1.1 Introducción

PSO es un método adaptativo que utiliza agentes o partículas que se mueven a través del espacio de búsqueda (véase Figuras 6.1 y 6.2) utilizando los principios de: Evaluación, Comparación e Imitación (c.f. [65], [66]).

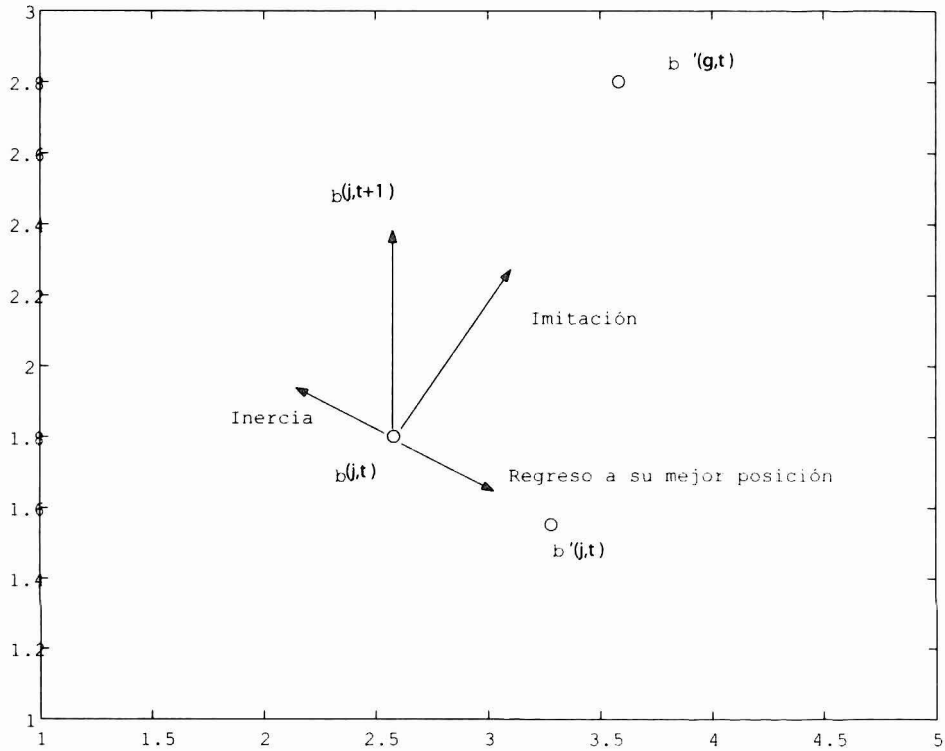


Figura 6.2: Posición de la i-ésima partícula y componentes de movimiento.

PSO se basa en el uso de un conjunto de partículas o agentes que corresponden a estados de un problema de optimización (véase Figura 6.1), donde cada partícula se moverá en el espacio de soluciones en busca de una posición óptima o una buena solución. En PSO los agentes se *comunican* entre sí, y entonces el agente con una mejor posición (medida de acuerdo a una función objetivo) *influye* en los demás atrayéndolos hacia él (véase figura 6.2).

La población se inicializa, asignando a las variables valores y una velocidad aleatorios. En cada iteración, la velocidad de cada partícula es aleatoriamente acelerada hacia su mejor posición (donde el valor de la función de aptitud u objetivo es mejor) y a través de las mejores posiciones de sus vecinos. En términos generales, como se mencionó anteriormente, las principales características de PSO son: evaluación, comparación e imitación.

La optimización por enjambre de partículas es una parte de lo que se

conoce como inteligencia de enjambre, tiene sus raíces en la vida artificial, psicología social, ingeniería y ciencia de la computación. PSO difiere de la computación evolutiva (c.f. [65], [66]) en que los miembros de la población llamados partículas o agentes, están *volando* a través del hiperespacio del problema.

1. Crear una población de partículas distribuidas en el espacio factible.
2. Evalúe cada posición de las partículas de acuerdo a una función objetivo (función de aptitud).
3. Si la posición actual de una partícula es mejor que las previas, actualícela.
4. Determine la mejor partícula (de acuerdo a las mejores posiciones previas).
5. Actualice las velocidades de las partículas $j = 1, 2, \ldots, n$, de acuerdo a:

$$V_{j,t+1} = \alpha V_{j,t} + rand(0, \varphi_1)[\beta'(j,t) - \beta(j,t)] +$$
$$+ rand(0, \varphi_2)[\beta'(g,t) - \beta(j,t)].$$

6. Mueva las partículas a sus nuevas posiciones de acuerdo a:

$$\beta_{j,t+1} = \beta_{j,t} + V_{j,t+1},$$

7. Vaya al paso 2 hasta que el criterio de finalización se satisfaga.

Figura 6.3: Algoritmo de PSO.

Las principales características de PSO son (c.f. [65], [66]):

- Evaluación: La tendencia al estímulo de evaluar, es la principal característica de los organismos vivos. El aprendizaje no ocurre a menos que el organismo pueda evaluar, pueda distinguir características del medio ambiente que atraen o características que repelen. Desde este punto de vista, el aprendizaje puede definirse como un cambio que posibilita al organismo mejorar la evaluación promedio de su medio ambiente.

- Comparación: Los estándares del comportamiento social se realizan mediante la comparación con otros.

- Imitación: Lorenz asegura que sólo los seres humanos y algunas aves son capaces de imitar. La imitación es central para la adquisición y mantenimiento de las habilidades mentales. Para mayor detalle se puede consultar [65], [66].

Para resolver problemas, se propone utilizar PSO con un manejo dinámico de las partículas, lo que permite romper ciclos y diversificar la búsqueda. Para resolver este problema, se considerará que un enjambre de n partículas-solución está dada de la forma:

$$\beta = (\beta_1, \beta_2, \ldots, \beta_n)$$

con $\beta_j \in \Omega, j = 1, 2, \ldots, n$, entonces definimos un movimiento del enjambre de la forma:

$$\beta_{t+1} = \beta_t + V_{t+1},$$

donde:

- Ω: Espacio factible de soluciones.

- $V_{j,t+1} = \alpha V_{j,t} + rand(0, \varphi_1)[\beta'(j,t) - \beta(j,t)] + rand(0, \varphi_2)[\beta'(g,t) - \beta(j,t)]$.

- $V_{j,t}$: Velocidad en el tiempo t de la j-ésima partícula.

- $V_{j,t+1}$: Velocidad en el tiempo t+1 de la j-ésima partícula.

- $\beta(j,t)$: Partícula j-ésima en el tiempo t.

- $\beta'(g,t)$: Partícula con el mejor valor de todas en el tiempo t.
- $\beta'(j,t)$: Partícula j-ésima con el mejor valor hasta el tiempo t.
- rand(0,φ): Valor aleatorio de distribución uniforme en el intervalo [0,φ].
- α: Parámetro de escala.

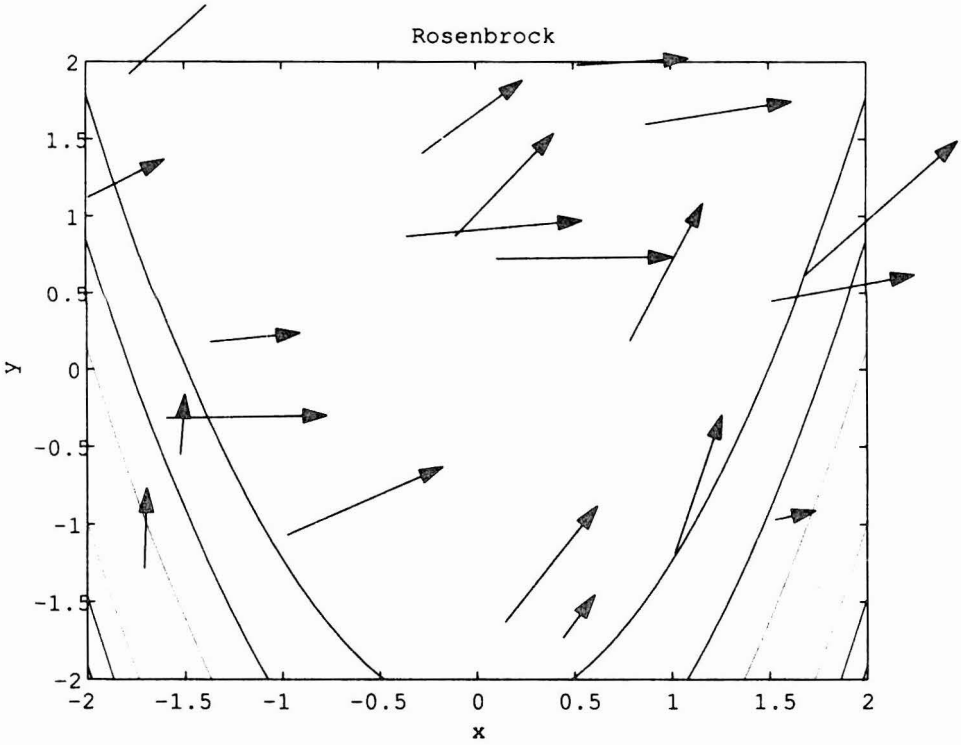


Figura 6.4: Curvas de nivel y vectores de velocidad del ejemplo.

6.1.2 Ejemplo: Optimización de la función de Rosenbrock

Aplicamos PSO a la minimización de la función de Rosenbrock en dos dimensiones, que habíamos introducido en la subsección 3.2.10.

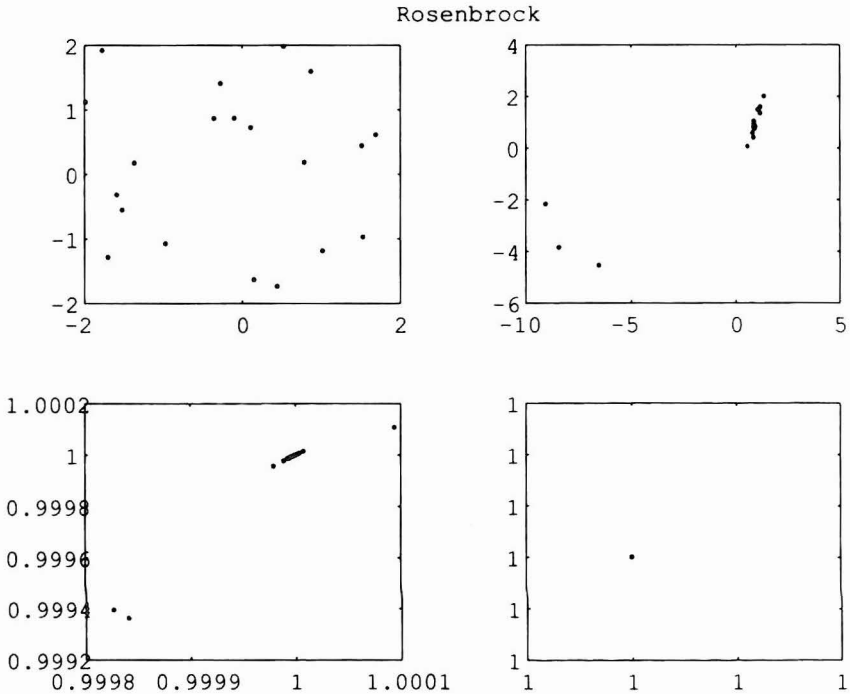


Figura 6.5: Población inicial y después de 100, 200 y 300 iteraciones del ejemplo.

Recordemos que la función tiene un valle largo y angosto y tiene un mínimo global de 0 en el punto (1,1).

Para el algoritmo de PSO, tomamos los siguientes valores para los parámetros:

Una población con 20 partículas, $\varphi_1 = \varphi_2 = 1.4$, y α variando linealmente desde 0.9 a 0.4 de acuerdo al número de iteraciones, que fijamos en un máximo de 300.

En la Figura 6.4 se muestran las curvas de nivel de la función, así como las posiciones de la población con los vectores de velocidad asociados. La dirección y longitud de cada vector de velocidad es su valor inicial; tanto la población como las velocidades han sido obtenidas de manera aleatoria.

En la Figura 6.5 se muestra en el cuadro superior izquierdo la población inicial, después en las iteraciones 100, 200, 300. Cada cuadro tiene una escala

diferente, y en el último (el cuadro inferior derecho), vemos que la población ya convergió al punto (1,1), que es el verdadero mínimo.

6.1.3 Ejemplo: Problemas de Inventarios

La apertura a los tratados internacionales ha forzando a los empresarios, quienes intentan mantener sus metas de competitividad, a mejorar la calidad de sus productos y servicios.

Muchas industrias de México y en el mundo operan con grandes montos de capital paralizado, históricamente, entre el 5% y el 40% de sus activos. Una forma de alcanzar una reducción en este costo es llevar un adecuado control en sus inventarios.

Inventarios multiproducto: Caso determinístico

.

El problema de inventarios multiproductos consiste, en general, en determinar cuánto y cuándo ordenar de cada artículo de los que se controlan, para que la demanda sea satisfecha con costos mínimos. En este trabajo, se supone que la entrega por parte de los proveedores es inmediata y se realiza en una sola entrega, no se permite déficit y los artículos nunca llegan a ser obsoletos una vez que se inventarean. Además, se supone que diferentes artículos de un mismo proveedor se pueden entregar en un solo paquete y se realiza a intervalos regulares de tiempo, digamos t_1.

El problema para n artículos por tanto, se traduce en determinar la frecuencia de los ciclos de abasto tanto individual como en conjunto, por lo que se puede observar fácilmente que el cociente de la frecuencia individual del artículo j digamos t_j entre la frecuencia conjunta es un entero positivo $\beta_j \geq 1$, es decir, $\beta_j = \frac{t_j}{t_1}$ para $j = 1, 2, \ldots, n$.

Los artículos tienen diferentes ciclos, pero el que marca las oportunidades para ordenar es el de menor duración t_1. Se ordena tantas veces como lo requiera dicho artículo en cada unidad de tiempo, t_j. Cada vez que se ordena se incluyen en la orden los artículos cuyo ciclo así lo indica.

Los costos que intervienen en el modelo son de: ordenación, adquisición e inventario. Un costo fijo K se carga cada vez que se realiza una orden. Un costo por línea k_j se produce cada vez que se carga el artículo j en la orden

correspondiente. El costo debido al inventario se carga sobre el inventario promedio durante un ciclo y está dado por: $CP = \frac{1}{2}(\sum_j h_j d_j t_j)$ donde: h_j, d_j y t_j son el inventario promedio, la demanda promedio y el tiempo del ciclo del artículo j, respectivamente. De lo anterior, se tiene que el costo total por unidad de tiempo es:

$$CT = \frac{1}{t_1}(K + \sum_j k_j\beta_j + \sum_j c_j d_j) + \frac{1}{2}t_1 \sum_j (h_j d_j \beta_j),$$

donde las c_j son los costos unitarios y lo que se desea es minimizar el costo total CT, el cual no es sólo función de t_1 sino también de las β_j y de las c_j, para $j = 1, 2, \ldots, n$. Una solución consiste en aproximar los valores de las β_j y después encontrar el valor de t_1 que minimiza a $CT(t_1)$.

A fin de iniciar la búsqueda para mejorar el plan maestro de abastecimiento, se requiere un punto inicial. Éste se obtiene de manera natural para cuando $\beta_j = 1, j = 1, 2, \ldots, n$ donde cada β_j está en cierto rango de valores, por lo que se llega a:

$$Minimizar \frac{1}{t_1}(K + \sum_j \frac{k_j}{\beta_j} + \sum_j c_j d_j) + \frac{1}{2}t_1 \sum_j h_j d_j \beta_j,$$

sujeto a: $t_1 > 0, \beta_j \in [1, \beta^*]$, y $\beta^* = 1, 2, \ldots, n$, éste es un problema de programación no lineal entero mixto, y la función de costo es convexa respecto a t_1, por lo que, una vez fijos los valores de las β_j, el valor óptimo para t_1 se encuentra fácilmente.

Para el ejemplo numérico propuesto, una partícula está representada por un arreglo 15-dimensional. Inicialmente se generan N partículas de la forma $\beta_j(1) = (\beta_{j1}, \beta_{j2}, \ldots, \beta_{j15})$, donde β_{ji} es un entero en el intervalo [1,15] $\forall i = 1, 2, \ldots, 15$, $j = 1, 2, \ldots, N$, y se procede con el algoritmo de PSO, sólo que al término de cada iteración se redondean al entero más cercano y cuidando que no salgan de los límites los valores de las β_{ji}, antes de realizar la evaluación.

Resultados Computacionales

Cabe señalar que la primer adaptación reportada de la técnica PSO al problema de inventarios multiproducto fue propuesta en [34]. Esta adaptación

probó su robustez para encontrar buenas soluciones al compararse contra algoritmos eficientes reportados en la literatura, en particular se compararon los siguientes:

Kaspi and Rodenblatt: (KyR) (c.f. [64]) , Búsqueda Tabú: BT (c.f. [14] y [39]), PSO: algoritmo propuesto.

Para probar el algoritmo propuesto se corrieron, para cada instancia, los tres algoritmos. Se generaron 100 instancias aleatorias para cada uno de los tamaños n = 5, 10, 15, 20, 25 y 30, con los siguientes rangos de las variables:

- $200 \leq dj \leq 50000$, j=1,2, ...,n,
- $1 \leq hj \leq 8$, j=1,2, ...,n ,
- $1 \leq kj \leq 12$, j=1,2, ...,n.

Se puede observar de la tabla que las eficiencias del algoritmo PSO son bastante buenas, con respecto al número de iteraciones que se realizaron fueron de $2n$ donde n es el número de artículos considerados. Cabe señalar que en [35] compararon la efectividad de varios algoritmos generando 132000 ejemplos y cubriendo 132 clases diferentes. En la Tabla 6.1 se presenta un ejemplo tomado de [35] donde se reportan los mejores resultados encontrados a través de los diferentes métodos.

Conclusiones y Perspectivas para el Ejemplo

En este ejemplo se enfocó la atención al análisis de la técnica de PSO, observando que esta técnica es aplicable a todo tipo de optimización en especial a problemas de programación combinatoria.

Se presentó un algoritmo de solución para el problema de inventarios multiproducto con demanda determinística mediante el esquema de PSO descubriendo buenas soluciones, aunque como se puede observar en la Tabla 6.1, PSO no proporciona el mejor resultado, se tiene que la codificación de PSO es mucho más sencilla y el tiempo de ejecución es mucho más rápido. Cabe mencionar que hay que realizar más investigación sobre la cantidad de partículas a utilizarse así como el de llevar un control sobre las vecindades de búsqueda y los demás parámetros de PSO.

j	d_j	h_j	k_j	KyR(β_j)	BT(β_j)	PSO(β_j)
1	10000	10	10	1	1	1
2	35000	2	8	1	1	1
3	8000	8	8	1	1	1
4	9000	5	6	1	1	1
5	50000	1	8	1	1	1
6	15000	2	5	1	1	1
7	20000	2	7	1	1	1
8	12500	1.6	4.5	1	1	1
9	10000	1.45	8	1	2	1
10	4250	2	5	2	2	2
11	2500	3	7	2	2	2
12	10000	1	10	1	2	2
13	4000	1.25	5	1	2	2
14	2000	3	6	1	2	2
15	1500	2	9	3	4	3
CT				1666	11419	11447

Tabla 6.1: Datos del ejemplo numérico tomado de [35].

6.1.4 Ejemplo: Regresión No Lineal.

En regresión no lineal, se encuentra el problema, tan frecuente en distintos métodos estadísticos, de la obtención de mínimos locales ([34], [76]). En efecto, el método clásico de regresión no lineal es el conocido como de Gauss-Newton, el cual está basado en una aproximación de la función de ajuste mediante el uso de un polinomio de Taylor de primer orden, y en un procedimiento de búsqueda de un óptimo local ([14], [34]). Por ello, es razonable pensar en la implementación de técnicas de optimización global. Entre estas técnicas, podemos citar el recocido simulado, la búsqueda tabú y los algoritmos genéticos [35] las cuales han sido ampliamente usadas en distintos problemas estadísticos, de investigación de operaciones o de ingeniería.

Todas estas implementaciones de las técnicas de optimización mencionadas han mejorado o al menos igualado los resultados conocidos, los detalles pueden consultarse en [33]. En regresión no lineal, ya se ha aplicado la técnica de recocido simulado y Búsqueda Tabú (ver [93], [94], [104], [107], [108], [110]) con resultados satisfactorios.

En la presente sección, proponemos un método estocástico para encontrar las soluciones iniciales así como una presentación de la regresión no lineal como problema de optimización, y describiremos algunos de los métodos más conocidos para llevarla a cabo. Enseguida se presenta la implementación efectuada de PSO para este problema tomado de [38]. Finalmente, damos algunas conclusiones y perspectivas a este ejemplo.

Conviene señalar que nos restringimos al caso de la regresión con una variable explicativa, pudiendo generalizarse fácilmente el trabajo al caso de varias variables explicativas; además, no abordamos los problemas relativos a la estimación estadística, como la obtención de intervalos de confianza para los parámetros.

Regresión No Lineal

Dadas dos variables x y y observadas sobre n objetos, donde x es una variable explicativa y y es una variable a explicar que depende de x, se quiere describir la relación de dependencia de y respecto a x mediante una función f; es decir, se quiere establecer la relación funcional $y = f(x) + \epsilon$, donde ϵ es un término de error (en este trabajo no supondremos que ϵ sigue alguna

distribución en particular). La función f depende generalmente de ciertos parámetros, cuyo vector denotaremos θ por lo que escribiremos a la función de regresión f_θ.

Se quiere minimizar el criterio de mínimos cuadrados, el cuál mide la calidad de la aproximación funcional propuesta:

$$S(\theta) = ||y - f(x)||^2 = \sum_{i=1}^{n} [y_i - f_\theta(x_i)]^2$$

donde $x = (x_1, \ldots, x_n)^t$ y $y = (y_1, \ldots, y_n)^t$ son los vectores de las observaciones de las variables, y $||\cdot||$ es la norma euclideana usual.

Salvo cuando f_θ es una función lineal, se conoce una solución general a este problema. Debe notarse que en algunos casos la función f_θ no es en sí lineal pero el problema de regresión se puede *linealizar*. Por ejemplo, el modelo $y = \theta_1 \exp(2x + \epsilon)$ es linealizable aplicando logaritmo natural, obteniéndose $y' = \theta_3 + \theta_2 x + \epsilon$, donde $y' = \ln y$ y $\theta_3 = \ln \theta_1$; sin embargo, el modelo $y = \theta_1 \exp(\theta_2 x) + \epsilon$ no es linealizable mediante la función logaritmo, ya que el error ϵ es aditivo.

En esta sección no nos ocuparemos de los modelos que son linealizables, ya que éstos se resuelven fácilmente mediante la regresión lineal clásica.

Algunos ejemplos de modelos que no son linealizables, son los siguientes:

- Crecimiento logístico: $y = \frac{\theta_1}{1+\exp[-\theta_2(x-\theta_3)]} + \epsilon$
- Crecimiento con decaimiento: $y = \theta_1 \exp(-\theta_2(x - \theta_3)^2) + \epsilon$
- $y = \frac{\theta_1}{\theta_1 - \theta_2}[\exp\{-\theta_2 x\} - \exp\{-\theta_1 x\}] + \epsilon$ (c.f. [34]).

Detalles de la Implementación

Se tomó cada partícula como un valor específico del vector de parámetros $\vec{\Theta}$, las que fueron generadas de manera aleatoria. Se consideraron $i = 1, \ldots, n$ partículas y, en cada iteración t, se identificaba la mejor posición de cada partícula $\vec{\Theta}_i^*(t)$ y la mejor partícula de todas en la iteración $\vec{\Theta}^*(t)$, esta evaluación se realizó mediante el criterio de mínimos cuadrados. Posteriormente se realizó el movimiento:

$$\vec{\Theta}_i^*(t+1) = \vec{\Theta}_i^*(t) + V(\vec{\Theta}_i^*(t+1))$$

,
donde

$$V(\vec{\Theta}_i^*(t+1)) = V(\vec{\Theta}_i^*(t)) + \gamma_1 * rand_1 * (\vec{\Theta}_i^*(t) - \vec{\Theta}_i(t)) + \gamma_2 * rand_2 * (\vec{\Theta}^*(t) - \vec{\Theta}_i(t)),$$

$rand_i$ son valores aleatorios con distribución uniforme [0,1] y $\gamma_1 + \gamma_2 \leq 4.2$.

Ejemplo numérico

Considérese el siguiente ejemplo dado en [10] y retomado en [108], en el que se quiere ajustar el modelo:

$$y = \beta_1 e^{-\beta_2 x},$$

para el conjunto de datos siguiente:

x	-2.5	-1	1	2
y	1	1.1	-1.1	0.2

La figura 6.6 muestran las curvas de nivel de la función para este conjunto de datos, y se puede apreciar la presencia de dos mínimos locales de $S(\vec{\beta})$, uno al frente y otro al fondo de la figura.

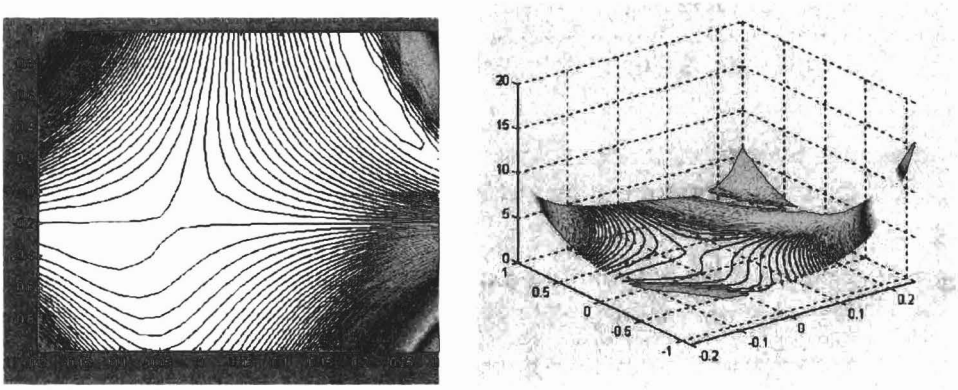


Figura 6.6: Ilustración de la presencia de mínimos locales.

Estos mínimos locales son $\vec{\beta}^* = (0.669, 0.214)$, con un valor de $S(\vec{\beta}^*) = 1.968$, y $\vec{\beta}^1 = (-0.764, -0.0298)$ con un valor de $S(\vec{\beta}^1) = 3.436$.

Resultados comparativos

Se obtuvieron los siguientes resultados (para mayor detalle vea [110]):

$$\beta^{*}=(\mathbf{0.669,0.214}),\ \beta^{1}=(\mathbf{-0.764,-0.0298}),$$

$$S(\beta^{*})=1.968,\ S(\beta^{1})=3.436$$

Recocido Simulado	98%
Búsqueda Tabú	92%
PSO	90%
Gauss-Newton	32%

Conclusiones para el ejemplo

Aunque el porcentaje es inferior a RS y BT, PSO no proporciona el mejor resultado, se tiene que la codificación de PSO es mucho más sencilla y el tiempo de ejecución es mucho más rápido. Cabe mencionar que hay que realizar más investigación sobre la cantidad de partículas a utilizarse así como el de llevar un control sobre las vecindades de búsqueda y los demás parámetros de PSO.

6.1.5 Ejercicios

1. Para cada uno de los problemas, realice las siguientes actividades:

   a Explique cómo sería una partícula del enjambre.

   b Proporcione una forma para crear el enjambre de partículas inicial.

   c Indique de manera explícita cómo evaluaría cada posición de las partículas.

   d Proporcione la manera de actualizar las velocidades de las partículas.

   e Explique cómo se realizaría el movimiento de las partículas a sus nuevas posiciones.

   f Proporcione un criterio de paro del algoritmo.

   g Haga un reporte con todos sus análisis y resultados.

I El problema de conjunto independiente, visto en la sección 3.1.4.

II El problema de coloración mínima, visto en la sección 3.1.5.

III El problema de empaquetamiento, visto en la sección 3.1.6.

IV El problema de cubrimiento por vértices, visto en la sección 3.1.7.

V El problema del agente viajero visto en la sección 3.1.8.

VI El problema de clan máximo, visto en la sección 3.1.9.

VII El problema de corte máximo, visto en la sección 3.1.10.

VIII El problema de satisfacibilidad, visto en la sección 3.1.11.

IX El problema de la mochila, visto en la sección 3.1.12.

2. Para cada uno de las siguientes funciones, vistas en la sección 3.2, realice las siguientes actividades:

a Explique cómo sería una partícula del enjambre.

b Proporcione una forma para crear el enjambre de partículas inicial.

c Indique de manera explícita cómo evaluaría cada posición de las partículas.

d Proporcione la manera de actualizar las velocidades de las partículas.

e Explique cómo se realizaría el movimiento de las partículas a sus nuevas posiciones.

f Proporcione un criterio de paro del algoritmo.

g Haga un reporte con todos sus análisis y resultados.

i. La función de Ackley

ii. La función RCOS de Branin

iii. La función de Easom

iv. La función de la esfera

v. La función de Goldstein-Price

vi. La función de Griewangk

vii. La función de Langermann

viii. La función de Michalewicz

ix. La función de Rastrigin
x. La función de Rosenbrock
xi. La función de Schaffer F6
xii. La función de Schwefel
xiii. La función de las seis jorobas del camello

3. Use el algoritmo de optimización por enjambre de partículas para resolver el problema del agente viajero con la matriz de costos no simétrica que aparece abajo.

a j =	1	2	3	4	5	6	7
de i = 1	×	13	9	19	22	3	7
2	6	×	7	8	39	4	8
3	18	3	×	5	18	4	3
4	4	16	6	×	5	13	16
5	14	22	99	17	×	8	10
6	3	21	7	3	10	×	9
7	11	19	8	19	8	3	×

6.2 Método Hormiga

Algunos algoritmos heurísticos se han propuesto basados en el comportamiento de las hormigas cuando éstas buscan su alimento. Las hormigas van dejando un rastro durante su trayecto en busca de alimento, de una substancia que secretan llamada feromona. Los algoritmos inspirados en los *sistemas de las hormigas* o *sistema hormiga* (AS del inglés *Ant System*) se han propuesto para resolver problemas de optimización combinatoria como son el problema del agente viajero y el problema de asignación cuadrático.

En esta sección se presenta el método hormiga. Se proporcionan las líneas generales de esta técnica basándose en lo propuesto en [6], [17], [40], [41], [42] y posteriormente se presentará una aplicación para el problema del agente viajero.

6.2.1 Introducción

Ant System, fue propuesto inicialmente por Dorigo [17] y [40], y se basa en el comportamiento colectivo de las hormigas en la búsqueda de sus alimentos.

Las hormigas tienen movimientos que son de tipo aleatorio puesto que estos animales son casi ciegos, pero a través de que van siguiendo el rastro de feromona dejado por sus compañeras, pueden seguir la misma trayectoria, desde su nido hasta la fuente de alimento, encontrada por sus predecesoras.

Las feromonas permiten a los animales tener una "comunicación" entre los de su misma especie, indicando por ejemplo, el sexo o su estado anímico, en el caso de las hormigas en su búsqueda de alimento, les permite seguir el rastro de sus compañeras, lo que permite que las hormigas individuales actúen como un todo después de cierto tiempo, permitiendo de esta manera encontrar las rutas más cortas desde su nido.

Para el caso del problema del agente viajero, por ejemplo, cada hormiga representa una solución que se va construyendo de la siguiente manera: inicialmente se pone una hormiga en una ciudad, y entonces, la hormiga agrega de manera probabilística una nueva ciudad y se continua el proceso hasta alcanzar la trayectoria completa. La probabilidad de que una hormiga siga un camino depende del número de hormigas y de la feromona que han dejado al utilizr un cierto camino, por lo que, una cantidad grande de feromonas en un camino significa una probabilidad alta de que se visitará.

Existen tres versiones de AS propuestas inicialmente por Marco Dorigo para el problema del agente viajero, que se diferencian en el momento y la manera de actualizar la matriz de feromonas, son:

1. *Ant-density*: con actualización de la matriz de feromonas de manera constante por donde pasa una hormiga,

2. *Ant-quantity*: con actualización de la matriz de feromonas inversamente proporcional a la distancia entre 2 ciudades recorridas, y

3. *Ant-cycle*: con actualización de la matriz de feromonas inversamente proporcional al trayecto completo, al terminar un recorrido.

6.2.2 Generalidades del Sistema Hormiga

En general, se puede considerar que la conducta de un nido de hormigas es de tipo colectivo, la cual está basada en la comunicación de sus miembros a través de la feromona secretada por éstos.

AS utiliza para el problema del agente viajero una representación gráfica (gráfo o gráfico de AS) donde cada arista (r, s) tiene cierta cantidad de feromona, la que se actualiza en cada iteración. De manera general, AS trabaja como sigue (c.f. [6], [17], [40], [41], [42]): cada hormiga genera una ruta completa escogiendo los nodos según una regla de transición de estado de tipo probabilístico dado por la ecuación 6.1; las hormigas prefieren moverse a nodos que se conectan por aristas de distancias cortas, que tienen una presencia de feromona alta. Una vez que todas las hormigas han completado sus rutas, una regla de actualización de la feromona global se aplica mediante la ecuación 6.2: una porción de la feromona se evapora en todas las aristas, entonces cada hormiga deposita una cantidad de feromona en las aristas que pertenecen a su ruta en la proporción de qué tan corta fue su ruta. De este modo, se continúa con una nueva iteración del proceso.

La ecuación 6.1 proporciona la probabilidad de la transición del nodo r al nodo s para la k-ésima hormiga:

$$P_{rs}^{k}(t) = \begin{cases} \dfrac{[\gamma_{rs}(t)]^{\alpha}[\eta_{rs}]^{\beta}}{\sum_{u \notin tabu_k(ru)}[\gamma_{ru}(t)]^{\alpha}[\eta_{ru}]^{\beta}} & \text{si} \quad s \notin tabu_k \\ 0 & \text{de otra forma} \end{cases} \tag{6.1}$$

Como cada hormiga k debe visitar todas las ciudades sólo una vez, se le asocia una estructura de datos llamada lista tabú de la hormiga, denotada como, $tabu_k$, que indica las ciudades que ya fueron visitadas por la hormiga. Esta lista tabú se inicializa una vez que la hormiga ha realizado un recorrido completo. Se define como $tabu_k(rs)$ al elemento de la lista tabú de la hormiga k que indica que pasó de la ciudad r a la ciudad s.

Además se tiene que:

$\gamma_{rs}(t)$ es la cantidad de feromona en la iteración t.

η_{rs} es el inverso de la distancia entre la ciudad r y la ciudad s.

β y α, son dos parámetros ajustables que determinan la importancia relativa de intensidad del sendero γ_{rs} versus la visibilidad η_{rs}. En AS, la regla de actualización global se lleva a cabo como sigue: una vez que todas las hormigas han construido sus rutas completas, la feromona (es decir, la intensidad de la ruta) se actualiza en todas las aristas según la ecuación 6.2:

Cd.	1	2	3	4	5	6
1	0	22.5	28.5	24.5	16.5	10.5
2	22.5	0	16.5	13.5	29.5	35.5
3	28.5	16.5	0	22.5	34.5	27.5
4	24.5	13.5	22.5	0	35.5	41.5
5	16.5	29.5	34.5	35.5	0	5.5
6	10.5	35.5	27.5	41.5	5.5	0

Tabla 6.2: Distancias entre ciudades para el ejemplo.

$$\gamma_{rs}(t) = (1-\rho)\gamma_{rs}(t-1) + \sum_{k=1}^{m} \Delta\gamma_{rs}^{k}(t) \tag{6.2}$$

Donde ρ es un coeficiente tal que $(1-\rho)$ representa la evaporación del sendero en una iteración (ruta), m es el número de hormigas, y $\Delta\gamma_{rs}^{k}(t)$ es la cantidad por unidad de longitud de la sustancia (feromona) de la ruta sobre la arista (r,s) por la hormiga k en esa iteración, teniéndose la ecuación 6.3.

$$\Delta\gamma_{rs}^{k}(t) = \begin{cases} \frac{1}{L_k(t)} & \text{si la arista (r,s)} \in \text{ruta completada por la hormiga } k \\ 0 & \text{de otra forma} \end{cases} \tag{6.3}$$

Donde $L_k(t)$ es la longitud de la ruta realizada por la hormiga k en la iteración t. Se actualiza la feromona poniendo una cantidad mayor de feromona a las rutas más cortas. En la Figura 6.7 se presenta un algoritmo general para AS (c.f.[6]).

6.2.3 Ejemplo: Agente Viajero.

Se tomará un ejemplo del problema del agente viajero para ver como se comporta el método hormiga. Considere la tabla 6.2 de distancias entre 6 ciudades. Inicialmente se puso una hormiga en cada ciudad y se dió un valor de uno para la feromona en cada arista.

Se realizaron varias corridas con diferentes valores iniciales para α, β y el factor de evaporización ρ, encontrándose los siguientes resultados:

Corrida 1.- $\alpha = 200$, $\beta = 2$ y $\rho = 0.95$

Ponga m hormigas al azar en los nodos de la gráfica de AS.

Repita hasta la convergencia del sistema o condición de paro.

Haga para i=1, hasta número de hormigas.

haga para $j = 1$, hasta número de ciudades.

escoge el nodo s para moverse, según la probabilidad de la transición, dada por la ecuación 6.1.

mueva la hormiga m al nodo s.

Actualice la feromona usando la fórmula de actualización de la feromona dada por la ecuación 6.2.

Actualice posición de las hormigas y condición de paro.

Vaya a paso 2.

Figura 6.7: Algoritmo General de AS.

Cd.	1	2	3	4	5	6
1	0	1.9500	1.9500	1.9500	1.9500	1.9500
2	1.9500	0	1.9500	1.9500	1.9500	1.9500
3	1.9500	1.9500	0	1.9500	1.9500	1.9500
4	1.9500	1.9500	1.9500	0	1.9500	1.9500
5	1.9500	1.9500	1.9500	1.9500	0	**6.9500**
6	1.9500	1.9500	1.9500	1.9500	2.9500	0
1	0	1.8525	1.8525	1.8525	1.8525	1.8525
2	1.8525	0	1.8525	1.8525	1.8525	1.8525
3	1.8525	1.8525	0	1.8525	1.8525	1.8525
4	1.8525	1.8525	1.8525	0	1.8525	1.8525
5	1.8525	1.8525	1.8525	1.8525	0	6.6025
6	**3.8525**	1.8525	1.8525	1.8525	6.8025	0
1	0	1.7599	1.7599	**1.7599**	1.7599	1.7599
2	1.7599	0	1.7599	4.7599	1.7599	1.7599
3	1.7599	1.7599	0	1.7599	1.7599	1.7599
4	1.7599	1.7599	1.7599	0	1.7599	1.7599
5	1.7599	1.7599	1.7599	1.7599	0	6.2724
6	6.6599	1.7599	1.7599	1.7599	6.4624	0
1	0	1.6719	1.6719	1.6719	1.6719	1.6719
2	1.6719	0	1.6719	6.5219	1.6719	1.6719
3	1.6719	1.6719	0	1.6719	1.6719	1.6719
4	1.6719	**5.6719**	1.6719	0	1.6719	1.6719
5	1.6719	1.6719	1.6719	1.6719	0	5.9588
6	6.3269	1.6719	1.6719	1.6719	6.1393	0

Tabla 6.3: Matrices de feromonas en la construcción de la ruta (5-6-1-4-2-3).

RUTA						COSTO
1	6	5	4	2	3	110
2	6	5	1	4	3	121
3	6	5	1	4	2	104
4	6	5	1	2	3	125
5	6	1	4	2	3	105
6	5	1	4	2	3	104

Tabla 6.4: Matriz de rutas y sus costos para la primera iteración de la corrida 1.

RUTA						COSTO
3	2	4	1	5	6	104
3	2	4	1	5	6	104
2	3	4	1	5	6	121
3	2	4	1	5	6	104
3	2	4	1	5	6	104
3	2	4	1	5	6	104

Tabla 6.5: Matriz de rutas y sus costos para la segunda iteración de la corrida 1.

Observe en la primer parte de la tabla 6.3, el valor más grande de feromona está en la arista (5,6), en la segunda parte de la tabla 6.3, para el nodo 6, el valor más grande de feromonas está en el arco (6,1), en la tercera parte, para el nodo 1, existe un empate, por lo que se tomó el arco (1,4). Para el nodo 4, el arco con mayor cantidad de feromona es (4,2) y, finalmente, dado que todos los nodos son tabú, a excepción del nodo 3, se completa la ruta con el arco (2,3).

En la segunda iteración para la corrida uno, se encontraron los datos de la tablas 6.5 y 6.6.

En la tercera iteración para la corrida uno, se encontraron los datos de la tablas 6.7 observándose una convergencia en las rutas a la óptima.

Ahora bien, no siempre se alcanza el valor óptimo, veamos un segundo ejemplo de corrida con valores de los parámetros $\alpha = 10$, $\beta = 0.5$ y $\rho = 0.95$, se presentan los resultados en las tablas 6.8, 6.9 y 6.10.

Se puede observar en las tablas 6.8, 6.9 y 6.10 que, dados los valores

Cd.	1	2	3	4	5	6
1	0	1.2290	1.2290	1.2290	1.2290	1.2290
2	1.2290	0	5.9124	9.9384	1.2290	1.2290
3	1.2290	1.2290	0	1.2290	1.2290	1.2290
4	1.2290	9.0157	1.2290	0	1.2290	1.2290
5	1.2290	1.2290	1.2290	1.2290	0 10.3802	
6	10.0658	1.2290	1.2290	1.2290	10.2129	0

Tabla 6.6: Matriz de feromonas en la construcción de la ruta (3-6-5-1-4-2).

RUTA						COSTO
6	5	1	4	2	3	104
6	5	1	4	2	3	104
6	5	1	4	2	3	104
6	5	1	4	2	3	104
6	5	1	4	2	3	104
6	5	1	4	2	3	104

Tabla 6.7: Matriz de rutas y sus costos para la tercera iteración de la corrida 1.

de los parámetros, se tiene una convergencia a una solución suboptimal en la tercera iteración, esto nos indica que, en general, hay que realizar varias corridas con diferentes valores de los parámetros.

Realicemos una última corrida, corrida 3, con valores de parámetros $\alpha = 2$, $\beta = 2$ y $\rho = 0.95$, se presentan los resultados desde la tabla 6.11 hasta la tabla 6.18.

Se puede observar en las tablas 6.15 a la 6.18 que se entra en un ciclo con rutas suboptimales.

Se puede observar que las tablas 6.4, 6.8 y 6.11 son iguales, esto se debe a que se tiene la misma rutina para encontrar las rutas de la primera iteración. Cabe destacar que, aunque se parte de las mismas rutas, depende de los valores de los parámetros para obtener las rutas de las demás iteraciones.

RUTA						COSTO
1	6	5	4	2	3	110
2	6	5	1	4	3	121
3	6	5	1	4	2	104
4	6	5	1	2	3	125
5	6	1	4	2	3	105
6	5	1	4	2	3	104

Tabla 6.8: Matriz de rutas y sus costos para la primera iteración de la corrida 2.

RUTA						COSTO
3	2	4	5	1	6	120
3	2	4	5	1	6	120
2	3	4	5	1	6	137
3	2	4	5	1	6	120
3	2	4	5	1	6	120
3	2	4	5	1	6	120

Tabla 6.9: Matriz de rutas y sus costos para la segunda iteración de la corrida 2.

RUTA						COSTO
6	1	5	4	2	3	120
6	1	5	4	2	3	120
6	1	5	4	2	3	120
6	1	5	4	2	3	120
6	1	5	4	2	3	120
6	1	5	4	2	3	120

Tabla 6.10: Matriz de rutas y sus costos para la tercera iteración de la corrida 2.

RUTA						COSTO
1	6	5	4	2	3	110
2	6	5	1	4	3	121
3	6	5	1	4	2	104
4	6	5	1	2	3	125
5	6	1	4	2	3	105
6	5	1	4	2	3	104

Tabla 6.11: Matriz de rutas y sus costos para la primera iteración de la corrida 3.

RUTA						COSTO
3	5	6	1	2	4	109
3	5	6	1	2	4	109
2	5	6	1	4	3	109
3	5	6	1	2	4	109
3	5	6	1	2	4	109
3	5	6	1	2	4	109

Tabla 6.12: Matriz de rutas y sus costos para la segunda iteración de la corrida 3.

RUTA						COSTO
4	6	5	1	2	3	125
4	6	5	1	2	3	125
3	6	5	1	4	2	104
4	6	5	1	2	3	125
4	6	5	1	2	3	125
4	6	5	1	2	3	125

Tabla 6.13: Matriz de rutas y sus costos para la tercera iteración de la corrida 3.

RUTA						COSTO
3	5	6	1	2	4	109
3	5	6	1	2	4	109
2	5	6	1	4	3	109
3	5	6	1	2	4	109
3	5	6	1	2	4	109
3	5	6	1	2	4	109

Tabla 6.14: Matriz de rutas y sus costos para la cuarta iteración de la corrida 3.

RUTA						COSTO
4	6	5	1	2	3	125
4	6	5	1	2	3	125
3	6	5	1	2	4	104
4	6	5	1	2	3	125
4	6	5	1	2	3	125
4	6	5	1	2	3	125

Tabla 6.15: Matriz de rutas y sus costos para la quinta iteración de la corrida 3.

RUTA						COSTO
3	5	6	1	2	4	109
3	5	6	1	2	4	109
4	5	6	1	2	3	113
3	5	6	1	2	4	109
3	5	6	1	2	4	109
3	5	6	1	2	4	109

Tabla 6.16: Matriz de rutas y sus costos para la sexta iteración de la corrida 3.

RUTA						COSTO
4	6	5	1	2	3	125
4	6	5	1	2	3	125
3	6	5	1	2	4	108
4	6	5	1	2	3	125
4	6	5	1	2	3	125
4	6	5	1	2	3	125

Tabla 6.17: Matriz de rutas y sus costos para la séptima iteración de la corrida 3.

RUTA						COSTO
3	5	6	1	2	4	109
3	5	6	1	2	4	109
4	5	6	1	2	3	113
3	5	6	1	2	4	109
3	5	6	1	2	4	109
3	5	6	1	2	4	109

Tabla 6.18: Matriz de rutas y sus costos para la octava iteración de la corrida 3.

6.2.4 Ejercicios

1. Considere el problema de asignación cuadrática dado en el capítulo 3:
   (a) Explique cómo sería una gráfica AS del problema (nodos y vértices).
   (b) Indique valores iniciales de α, β y ρ.
   (c) Proporcione una explicación detallada de cómo se movera una hormiga de un vértice a otro.
   (d) Indique cómo construir la tabla tabú de las hormigas.
   (e) Explique qué sería la distancia entre nodos.
2. Considere el problema de la mochila dado en el capítulo 3:
   (a) Explique cómo sería una gráfica AS del problema (nodos y vértices).
   (b) Indique valores iniciales de α, β y ρ.
   (c) Proporcione una explicación detallada de cómo se movera una hormiga de un vértice a otro.
   (d) Indique cómo construir la tabla tabú de las hormigas.
   (e) Explique qué sería la distancia entre nodos.

Capítulo 7

Algoritmos Genéticos y Estrategias Evolutivas

7.1 Introducción

Los algoritmos evolutivos son un conjunto de algoritmos basados en el proceso de la evolución natural. Existen diferentes tipos de algoritmos evolutivos los cuales se conocen como *algoritmos genéticos*, *programación evolutiva*, *estrategias evolutivas* o *programación genética*, el último es un descendiente de los algoritmos genéticos [92]. La diferencia entre los algoritmos evolutivos puede estar definida en las representaciones o formas de codificación, esquemas de selección y operadores de búsqueda; por ejemplo, los algoritmos genéticos (AG), por lo general, usan como operadores de búsqueda el cruzamiento como principal operador y como secundario a la mutación, aunque a veces suele sólo aplicarse el cruzamiento, mientras que las estrategias evolutivas (ES) generalmente usan mutación. La respuesta sobre cuál algoritmo es el mejor para ser aplicado depende del problema a resolver [111].

Existen dos características que distinguen a estos algoritmos de otros algoritmos computacionales de búsqueda no basados en la naturaleza:

1. Que todos ellos se basan en una población inicial, misma que representa a un conjunto de soluciones de un problema.

2. Que existe comunicación e intercambio de información entre los individuos de la población; ésta es el resultado de los operadores de búsqueda

aplicados a los individuos. A estos operadores de búsqueda se les conoce como operadores genéticos [111].

7.2 Descripción General de los AG y la ES

Los algoritmos genéticos fueron desarrollados por John Holland, quien quería construir un esquema teórico para la adaptación basada en la naturaleza. Darwin había postulado en su libro "El Origen de las Especies" que:

1. Existen variaciones en los rasgos de los individuos de cada especie.
2. Algunos rasgos son heredados.
3. Únicamente algunos individuos sobreviven lo suficiente para reproducirse; estos son elegidos naturalmente para propagar sus rasgos.

Pero, ¿cómo incorporarlos en un programa de computadora? A fines de la década de 1960, Holland desarrolló una técnica que permitió su incorporación, y se hizo popular bajo el nombre *algoritmo genético* tras la publicación de su libro en 1975 [26]. El concepto detrás de los algoritmos genéticos es la evolución natural; estos algoritmos buscan modelar algunos fenómenos naturales como la herencia genética y la lucha por la supervivencia. La lucha por la supervivencia implica que cada especie se adapta de la mejor manera a su medio ambiente, el cual va cambiando con el tiempo haciéndose cada vez más complejo. La herencia genética que cada especie tiene, debe mejorar de una generación a otra, ya que se deben desarrollar características que permitan a la nueva especie una mejor adaptación a su medio ambiente para que pueda sobrevivir en él. Esta herencia se encuentra en los cromosomas de la especie. Por ejemplo, en el reino animal, hay especies cuyo color de piel ha ido evolucionando con el tiempo, de tal forma que se mimetizan en el medio ambiente en el que viven y pasan desapercibidos para otras especies que podrían devorarlos, o bien crean nuevos mecanismos de defensa para su supervivencia [80].

Otras técnicas muy semejantes a los algoritmos genéticos son las llamadas estrategias evolutivas. Las estrategias evolutivas fueron desarrolladas por

Bienert, Rechenberg, y Schwefel en la década de los 1960 en Alemania, en trabajos dirigidos a problemas de optimización numérica.

La importancia de los AG y ES radica en que existe una serie de problemas interesantes, como los problemas NP mencionados en el capítulo 3, para los cuales los algoritmos tradicionales no representan soluciones factibles; muchos son problemas de optimización y métodos como los algoritmos genéticos o las estrategias evolutivas han sido aplicados dando buenos resultados [63][97].

La utilización de estas técnicas se basa en la idea de fijar el objetivo del problema mediante una función matemática, denominada función de costo o función objetivo, la cual hay que optimizar; en caso de maximizarla, valores menores de la función representan a las soluciones peores del problema.

7.3 Algoritmos Genéticos

Como se mencionó anteriormente, los algoritmos genéticos se basan en los principios de la evolución mediante la selección natural y toman un vocabulario semejante a la genética natural.

Inicialmente, tenemos una función objetivo que queremos optimizar sobre algún dominio. Los elementos del dominio representan soluciones potenciales al problema que se desea resolver, y en los AG es usual codificar estos elementos por medio de una representación binaria de la forma $b = a_1 a_2 \ldots a_K$, donde cada a_i tiene el valor 0 o 1. A las representaciones binarias se les llama individuos, cromosomas, o genotipos, y a los bits en la cadena, genes. La representación binaria de un cromosoma facilita la aplicación de los operadores genéticos, y proporciona un esquema general para atacar una variedad de problemas. Sin embargo, esta representación tiene sus desventajas cuando es aplicada a problemas numéricos de alta precisión, donde la precisión determina el número de decimales utilizados. La determinación de esta precisión fija la longitud del cromosoma que representa.

Se habla de la aptitud de un cromosoma, y de la función de aptitud, cuando los valores de esta función corresponden a los valores de la función objetivo sobre el elemento correspondiente al cromosoma. De esta forma, la aptitud indica que tan bueno es cada cromosoma como solución al problema,

e incluso, usamos el término de la función de aptitud definida sobre el espacio de los cromosomas.

En los AG, se emplea una población formada por un conjunto de cromosomas. A esta población se aplican los operadores genéticos para realizar una búsqueda en el espacio de los cromosomas para encontrar el óptimo al problema. Tal búsqueda requiere balancear dos objetivos:

1. Explorar alrededor de la mejor solución encontrado.
2. Explorar el espacio de búsqueda.

En resumen, un algoritmo inicial para un AG consiste en:

1. Codificación del problema como una cadena binaria.
2. Generación de una población.
3. Aplicación repetida de los operadores genéticos hasta encontrar una solución.

En el resto del capítulo, vamos a suponer que la tarea es la de maximizar la función objetivo. En caso contrario, podemos maximizar el valor negativo de la función objetivo.

A continuación, describimos los operadores genéticos.

7.3.1 Operadores Genéticos

Los operadores genéticos que se aplican en estos algoritmos son: selección, cruzamiento y mutación. En seguida, se explicarán estos operadores de forma general ya que, dependiendo del problema, a veces se hacen algunas variaciones importantes.

Métodos de Selección

Los métodos de selección se basan en escoger que individuos tienen más oportunidades a ser elegidos a reproducirse dentro de la población, se esperaría que los elegidos fueran los más aptos; sin embargo, al igual que en la naturaleza, los individuos menos aptos también tienen la posibilidad de reproducirse, no obstante, esta posibilidad debe ser menor a la de los más aptos.

El método más popular de la reproducción es similar al comportamiento de una rueda de ruleta, donde cada ranura tiene tamaños diferentes proporcionales al valor de aptitud, es decir, de acuerdo a su aptitud. La técnica es llamada el *método de la ruleta* y selecciona el candidato a ser reproducido por selección proporcional. El método requiere que la función de aptitud no tenga valores negativos. La técnica puede ser implementada mediante un algoritmo como se describe a continuación:

1. Se calculan las aptitudes de todos los miembros de la población y se encuentra la suma, llamada la *aptitud total.* Matemáticamente, si la población de cadenas binarias es $\{b_1, \ldots, b_n\}$ y la función de aptitud es f, entonces la aptitud del miembro b_i es $f(b_i)$, y la aptitud total es:

$$\text{aptitud total} = \sum_{i=1}^{n} f(b_i)$$

2. Se calculan las probabilidades, p_m, de elegir cada miembro de la población:

$$p_m = \frac{f(b_m)}{\sum_{i=1}^{n} f(b_i)}$$

3. Se calculan las probabilidades acumuladas, q_m, por cada miembro de la población:

$$q_m = \sum_{i=1}^{m} p_i$$

4. Se genera r, un número aleatorio entre 0.0 y 1.0.

5. El algoritmo regresa el primer miembro de la población cuya probabilidad acumulada sea mayor o igual al número r aleatorio, es decir, se escoge b_m si: $q_{m-1} < r \leq q_m$

6. Se repiten los pasos del algoritmo n veces para encontrar la nueva población, análoga a dar n vueltas a la rueda de ruleta.

En el siguiente ejemplo se ilustra la técnica de la ruleta. Considere la población formada con los seis cromosomas:

$$\{01110,\ 11000,\ 00100,\ 10000,\ 01101,\ 00011\}$$

,
donde la aptitud de cada cromosoma esta mostrado en la columna etiquetada 'Aptitud' de la Tabla 7.1. Se calculan los porcentajes en la siguiente columna para cada cromosoma, mediante la fórmula:

$$(\text{aptitud}/\text{aptitud total}) * 100,$$

donde la aptitud total es igual a 10.34; se muestra los correspondientes porcentajes en la ruleta de la figura 7.1.

La probabilidad acumulada, en la siguiente columna, se calculó acumulando las probabilidades de cada miembro, el cromosoma 1 tiene aptitud igual a 2.17, por lo tanto su probabilidad acumulada es de $0.2100 = 2.17/10.34$, el cromosoma 2 tiene aptitud igual a 1.3 más 2.17 del primero da su aptitud acumulada de 3.47 y una probabilidad acumulada igual a $0.3357 = 3.47/10.34$, el cromosoma 3 tiene aptitud 1.3 más 3.47 de los primeros dos, dando una aptitud acumulada de 4.77 y una probabilidad acumulada igual a $0.4614 = 4.77/10.34$, etcétera.

Se generan seis números aleatorios entre 0.0 y 1.0 y se escoge el primer cromosoma que se encuentra con una probabilidad acumulada igual o mayor al generado aleatoriamente. Los miembros con la aptitud más alta tendrán mayor posibilidad de sobrevivir. En la tabla 7.2 se encuentran los datos del ejemplo, así como los cromosomas que sobrevivieron para formar parte de la nueva población en la siguiente generación.

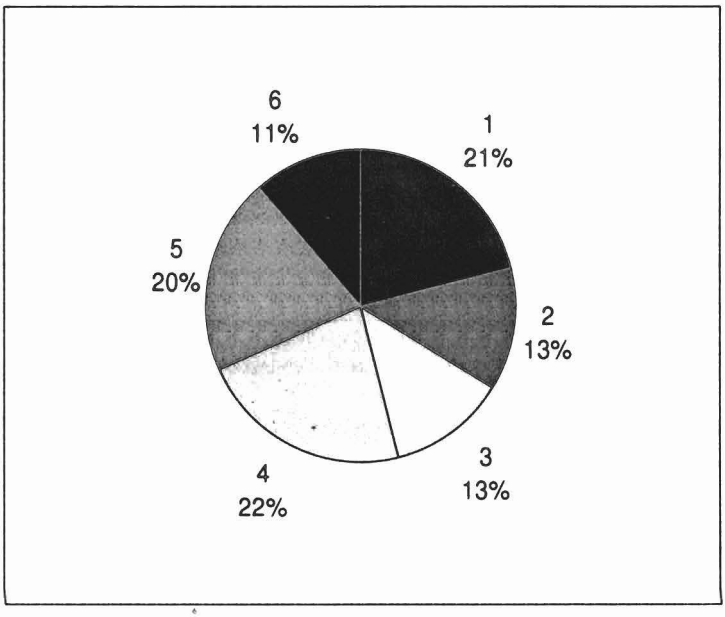


Figura 7.1: Método de selección con la ruleta.

Cruzamiento

El cruzamiento simula la descendencia, ambos padres heredan genes a su hijo y el cruzamiento simula este proceso; es un mecanismo de recombinación. Se seleccionan dos padres, sin reemplazo, por medio de una probabilidad de cruzamiento se decide si se efectúa el cruzamiento. Esto significa que no todas las parejas de cromosomas se cruzarán, sino que habrá algunas que pasarán intactas a la siguiente generación. En caso de que así sea, el sitio de cruzamiento es seleccionado de forma aleatoria y las colas de los cromosomas a partir del sitio de cruzamiento están intercambiadas, como en la figura 7.2, para formar dos nuevos cromosomas.

En este ejemplo se observa que el sitio de cruzamiento fue el 4, donde la parte terminal del primer cromosoma es ahora la parte terminal del segundo, y la parte terminal del segundo cromosoma pasa a ser ahora la parte terminal del primero.

La técnica de cruzamiento puede ser implementada mediante un algoritmo como se describe a continuación:

1. Se fija una probabilidad de cruzamiento, P_c, desde el principio.

No. de cromosoma	Cadena cromosómica	Aptitud	% con respecto al total	Probabilidad acumulada
1	01110	2.17	20.99	0.2100
2	11000	1.30	12.57	0.3357
3	00100	1.30	12.57	0.4614
4	10000	2.30	22.25	0.6839
5	01101	2.10	20.34	0.8872
6	00011	1.17	11.28	1.0000

Tabla 7.1: Datos del resultado del ejemplo.

Números aleatorios	Cromosomas sobreviv.
0.0185	1
0.8214	5
0.4447	3
0.6154	4
0.7919	5
0.9218	6

Tabla 7.2: Resultados después de la primera generación.

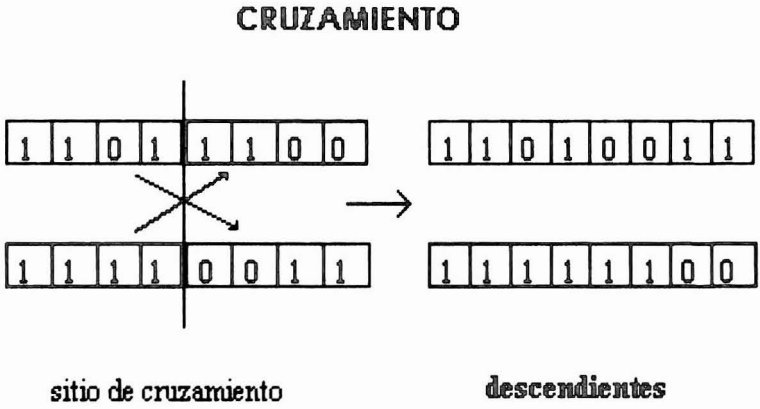


Figura 7.2: Ejemplo de un solo punto de cruzamiento.

2. Se seleccionan parejas de cromosomas de la población, sin reemplazo.

3. Se genera r, un número aleatorio entre 0.0 y 1.0.

4. Si $r \leq P_c$, se efectúa el cruzamiento.

5. Se genera un entero n aleatorio entre 1 y K-1, donde K es la longitud de los cromosomas, como el sitio de cruzamiento.

6. A partir de n, se intercambian las colas de los cromosomas para formar dos cromosomas nuevos.

Existen otros tipos de cruzamiento, por ejemplo, 2 puntos de cruzamiento. Cuando se usan 2 puntos de cruzamiento, se procede de manera similar, pero en este caso el intercambio se realiza en la forma mostrada en la figura 7.3.

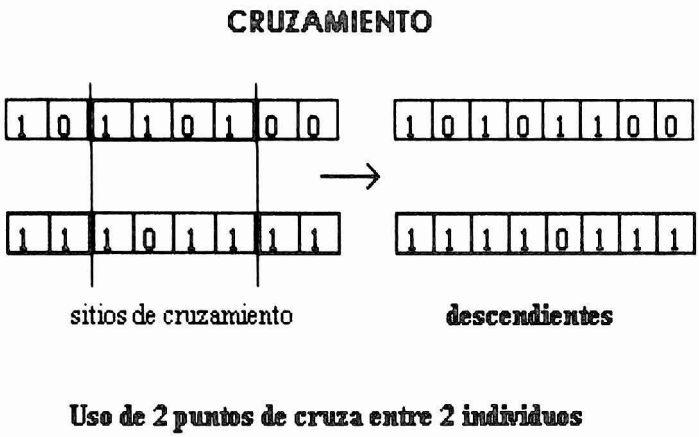


Figura 7.3: Cruzamiento con dos puntos.

En la figura 7.3 se usaron 2 puntos de cruzamiento entre 2 individuos. En este caso se mantienen los genes de los extremos y se intercambian los del centro. También aquí existe la posibilidad de que uno o ambos puntos de cruzamiento se encuentren en los extremos de la cadena, en cuyo caso se hará un cruzamiento usando un solo punto o ningún cruzamiento, según corresponda.

Mutación

Además de la selección y el cruzamiento, existe otro operador genético llamado mutación. Éste opera de manera independiente sobre cada cromosoma;

por medio de una probabilidad de mutación puede realizar un cambio a cada uno de los genes del cromosoma. Cuando se usa una representación binaria, un bit se sustituye por su complemento (un cero cambia en uno y viceversa). Este operador permite la introducción de nuevo material cromosómico en la población, tal y como sucede con sus equivalentes biológicos.

La técnica de mutación puede ser implementada mediante un algoritmo como se describe a continuación:

1. Se fija una probabilidad de mutación, P_m, desde el principio.
2. Se seleccionan un cromosoma de la población.
3. Por cada gen del cromosoma, se genera r, un número aleatorio entre 0.0 y 1.0.
4. Si $r \leq P_m$ se cambia el gen.

Por ejemplo, si $P_m = 0.1$ y tenemos el cromosoma 01101, luego generamos cinco números aleatorios 0.31, 0.01, 0.92, 0.33, 0.78 y revisamos cuales tienen valor menor o igual a P_m; en este caso, el segundo número aleatorio únicamente es menor, de modo que cambiamos solamente el segundo bit en el cromosoma para obtener el nuevo cromosoma 00101.

Se ha investigado la importancia de cruzamiento y mutación para los algoritmos genéticos. Varios investigadores han enfatizado la importancia de cruzamiento sobre mutación; sin embargo, no hay una demostración definitiva [58] y puede ser que depende del problema bajo consideración. Resultados empíricos (c.f. [55]) sugieran valores para P_c de más de 0.6, y para P_m menos de 0.1 (lo cual es diferente para el caso de las estrategias evolutivas).

7.3.2 Algoritmo Genético Básico

Podemos describir el funcionamiento básico de un algoritmo genético. Inicialmente, hay que fijar el tamaño de las cadenas binarias de los cromosomas, así como el número de cromosomas en la población. Esto depende del problema y del dominio de la función objetivo. Es usual generar los cromosomas en la población inicial de manera aleatoria después hay que escoger los valores para las probabilidades de cruzamiento y mutación; finalmente, un criterio

para la terminación del algoritmo es necesario: algunas posibilidades son, por ejemplo, un número fijo de iteraciones o continuar hasta que no haya una mejora sustancial en las soluciones obtenidas. El algoritmo está representado en la figura 7.4, donde se forma la población inicial y evalúa cada cromosoma. Después entra en un ciclo con las operaciones sucesivas de: selección, cruzamiento, mutación y evaluación de la nueva población. La evaluación consiste en encontrar la aptitud de cada cromosoma de la población. Si el criterio de terminación es logrado, el algoritmo termina y devuelve la mejor solución en la población; si no, se regresa al paso de selección.

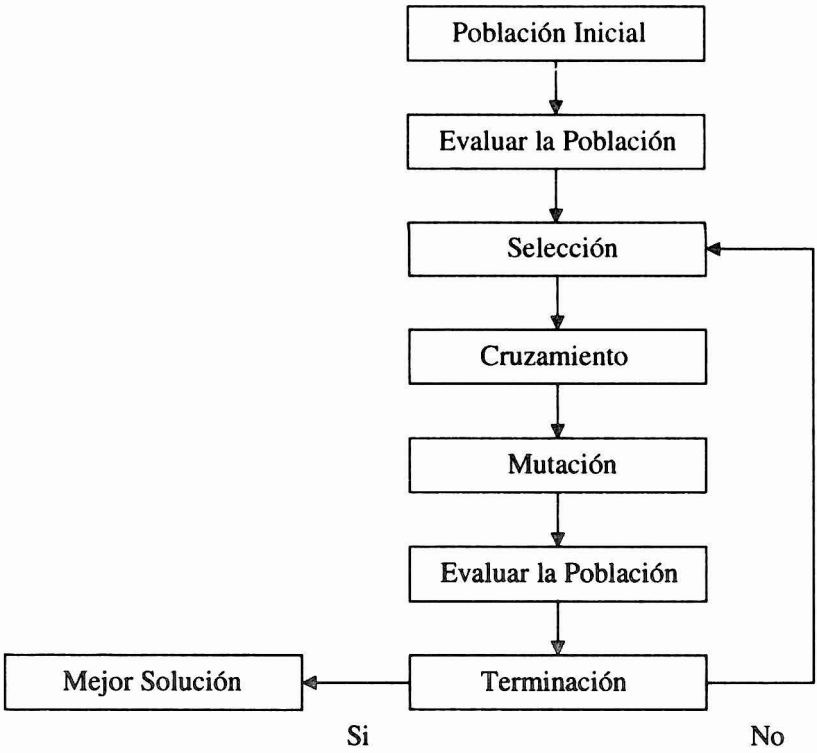


Figura 7.4: Un algoritmo genético.

7.3.3 Elitismo

Cuando aplicamos el algoritmo genético señalado anteriormente en la figura 7.4 a un problema específico, nos gustaría tener la confianza de que después

Los parámetros de entrada son: tamaño de la población, probabilidades de cruzamiento P_c y de mutación P_m y condición para la terminación.

Comienza

inicializa la población

evalúa los cromosomas de la población

repetir

comienza

selecciona los cromosomas en la nueva población
aplica cruzamiento a los cromosomas de la población
aplica mutación a los cromosomas de la población
evalúa los cromosomas de la población

termina

hasta condición de terminación

devuelve la solución más apta en la población

Termina

Figura 7.5: Pseudocódigo de un algoritmo genético.

de un número finito de pasos, obtenemos la solución óptima al problema, independientemente de la inicialización del algoritmo. Además, si después el algoritmo siempre mantiene la solución en la población, decimos que el algoritmo converge al óptimo. Rudolph [95], mostró, usando Cadenas de Markov, que los algoritmos genéticos de la figura 7.4 no convergen a un óptimo global, a menos que vayan guardando al mejor individuo de entre todas las generaciones, al usar una estrategia llamada elitista. La forma en que selecciona a dicho individuo es la siguiente: en la primera generación, basado en la función de aptitud, se guarda al mejor individuo; en la siguiente generación encuentra al mejor y al peor individuo; si el mejor de esta generación supera al ya seleccionado, se reemplaza al anterior individuo por éste, en caso con-

trario, remplaza el peor individuo de la población actual con el mejor de la generación anterior; este proceso se aplica el número máximo de generaciones definidas por el usuario.

7.3.4 Ejemplo: Optimización de funciones reales

Vamos a aplicar un algoritmo genético para calcular el valor máximo de la siguiente función en el intervalo [0,20]:

$$f(x) = x + 5 * sin(3 * x) + 8 * cos(5 * x).$$

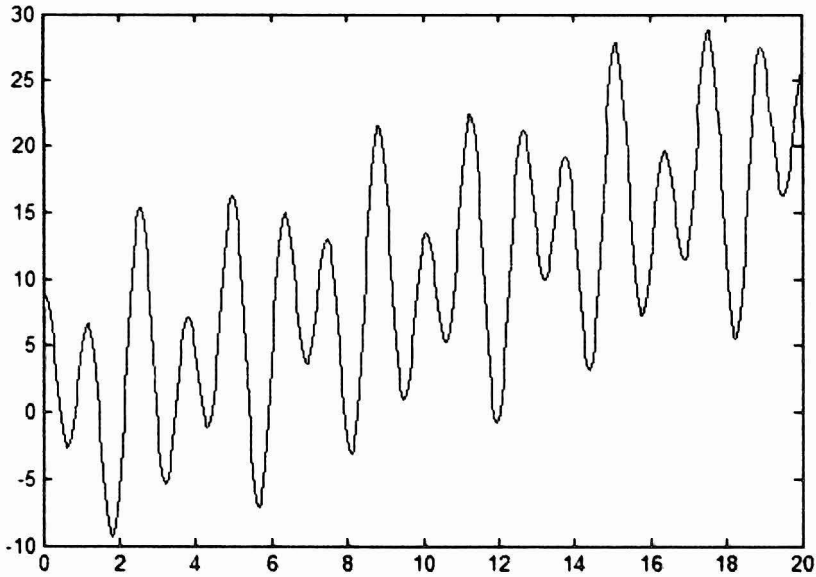


Figura 7.6: $f(x) = x + 5 * sin(3 * x) + 8 * cos(5 * x)$.

En general, si la función no tuviera una derivada conocida o fuera no continua, los algoritmos genéticos son una herramienta útil para encontrar los óptimos de funciones con estas características, donde las técnicas del análisis tradicional no se pueden aplicar; por lo tanto, para funciones sin características como las ya mencionadas, los algoritmos genéticos funcionan adecuadamente.

Se muestra la gráfica de la función en la figura 7.6. Como se puede apreciar, la función tiene varios máximos locales en el rango [0,20], y el máximo global se encuentra entre 17 y 18. Más aún, como tiene valores negativos en el intervalo, no podemos aplicar el método de la ruleta directamente. Para evitar estos problemas, primero vamos a encontrar el máximo de la siguiente función:

$$g(x) = x + 5 * sin(3 * x) + 8 * cos(5 * x) + 10.$$

Vamos a ver los pasos involucrados para aplicar un algoritmo genético a este problema:

1. Para aplicar un algoritmo genético, primero se decide como representar los cromosomas, como cadenas binarias. En general, si la cadena binaria $z = a_1 a_2 \ldots a_K$, representa un número real, x, en el intervalo $[a, b]$, se puede encontrar x mediante los siguientes pasos:

   (a) Se encuentra el valor decimal de z, $dec(z)$ por:

   $$dec(z) = \sum_{i=1}^{K} 2^{K-i} a_i$$

   $$x = a + dec(z) * \frac{(b-a)}{(2^K - 1)}.$$

   Por ejemplo, si $z = 1101$, entonces $dec(z) = 2^3 * 1 + 2^2 * 1 + 2^1 * 0 + 2^0 * 1 = 13$.

   La longitud K, de la cadena binaria, representa la precisión con la cual podemos representar los números reales del intervalo: para cadenas más largas obtenemos una precisión mayor. Aquí, las posibles soluciones son los números reales en el intervalo [0,20], y si se quiere obtener una precisión de 6 dígitos después del punto decimal, se tendrá que dividir el intervalo [0,20] en 20*1,000,000 tamaños de intervalos iguales; esto significa que son necesarios 25 bits para representar a cada cromosoma, es decir:

   $$16,777,216 = 2^{24} < 20,000,000 \leq 33,554,432 = 2^{25}.$$

Por ejemplo, el cromosoma 1000101110110101000111001 representa el número decimal 18,311,737, y $x = 0 + \frac{18,311,737*20}{(2^{25}-1)} = 10.914646$.

2. Ahora que se tiene la representación para los cromosomas, hay que escoger los parámetros para: el tamaño de la población, el criterio de terminación y las probabilidades de cruzamiento y mutación. Usualmente, este paso requiere cierta experimentación con los parámetros. Aquí vamos sencillamente a tomar una población con 20 cromosomas, 50 iteraciones, $P_c = 0.8$, y $P_m = 0.1$. Además, se va a incluir elitismo.

3. Primero se inicializa la población con cromosomas de cadenas binarias de tamaño de 25 bits definidos de forma aleatoria. Los puntos señalados en la figura 7.7 es la población inicial generada de forma aleatoria (20 puntos).

4. Se evalúa cada cromosoma, z, de la siguiente forma para encontrar la aptitud asociada, $eval(z)$: sea x el número real determinado a partir de z en (2) de arriba, entonces:

$$eval(z) = g(x).$$

Por ejemplo, si z = 1000101110110101000111001, entonces:

$$x = 10.914646, \text{y } eval(z) = g(x) = 22.618494.$$

Al maximizar la función, se busca al cromosoma con la aptitud más alto.

5. Se aplican los operadores genéticos de selección, cruzamiento y mutación. En caso de selección, se usa el método de la ruleta, y los otros dos operadores están como se describierón anteriormente. Se incluye elitismo.

Por ejemplo, en caso de cruzamiento, si tenemos los cromosomas v_1 y v_2, y el punto de cruzamiento fué seleccionado como el quinto gen:

$$v_1 = (11110|00011001100101010000),$$

$$v_2 = (10000|00000011000001101111),$$

el resultado del cruzamiento es:

$$v_1' = (11110|00000011000001101111),$$
$$v_2' = (10000|00011001100101010000),$$

Los números reales correspondientes a v_1, v_2, v_1', v_2' son:

18.812456, 10.007390, 18.7573909, 10.0624564, respectivamente. Al evaluar se tiene:

$$eval(v_1) = 36.119875,$$
$$eval(v_2) = 22.877551,$$
$$eval(v_1') = 34.5579516,$$
$$eval(v_2') = 23.3436692,$$

mejorando el cromosoma v_2.

En caso de mutación, por ejemplo, suponiendo que se seleccionó el segundo gen del cromosoma $v_1 = (111100001100110010101000 0)$ para ser mutado, éste, por ser igual a 1, se muta cambiando a 0, así:

$$v_1'' = (101100001100110010101000 0),$$

y éste cromosoma representa el valor $eval(v_1'') = 28.99189172$.

En la figura 7.8, se muestran con un círculo los valores obtenidos en la primera generación. Los otros puntos siguen siendo los iniciales graficados en la figura 7.7.

En la figura 7.9, se dan los valores de los mejores individuos por cada generación.

Finalmente, el valor máximo de la función $g(x) = x + 5 * sin(3 * x) + 8 * cos(5 * x) + 10$ fué encontrado en: $x = 17.5439$ donde $g(x) = 38.804736$ y, por lo tanto, $f(x)$= 28.804736, que se obtuvo en la generación 24, conservándose hasta la 50.

En la figura 7.10 se grafica el valor máximo encontrado.

Ahora vemos como se puede extender la codificación binaria a funciones multivariadas.

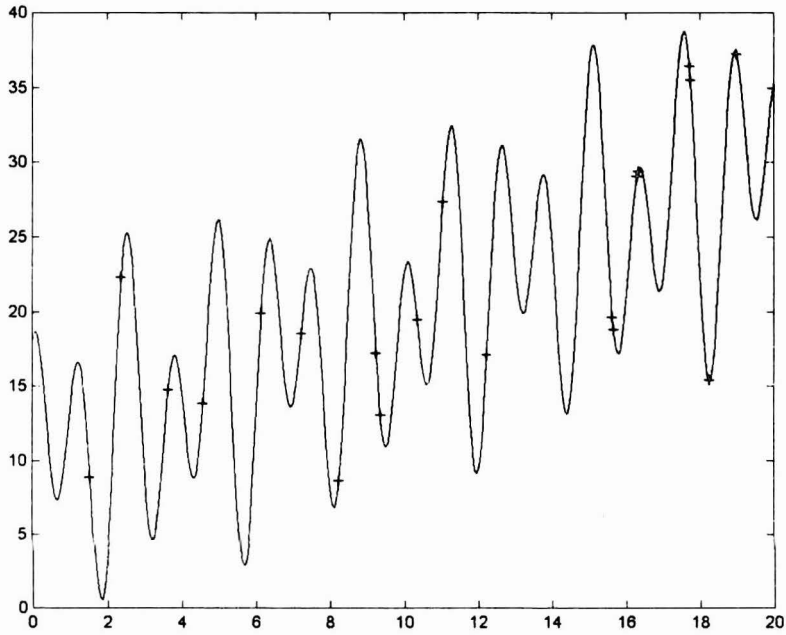


Figura 7.7: Población inicial.

7.3.5 Ejemplo: Optimización de funciones reales de varias variables

Vamos a aplicar un algoritmo genético para calcular el valor máximo de la siguiente función en el cuadrado [0,20] x [0,20] del plano:

$$f(x,y) = [x + 5 * sin(3 * x) + 8 * cos(5 * x) + y + 5 * sin(3 * y) + 8 * cos(5 * y)]/2$$

Se muestra la gráfica de la función en la figura 7.11; como se puede apreciar, la función tiene varios máximos locales en el rango [0,20] x [0,20] y sabemos, por el Ejemplo 1, que el máximo global es de 28.804736 se encontró en (17.5439, 17.5439).

Si requiere de varias variables para describir un cromosoma, podemos codificar cada una de éstas como una cadena binaria aislada y luego concatenarlas para producir el cromosoma. En el ejemplo actual, hay dos variables,

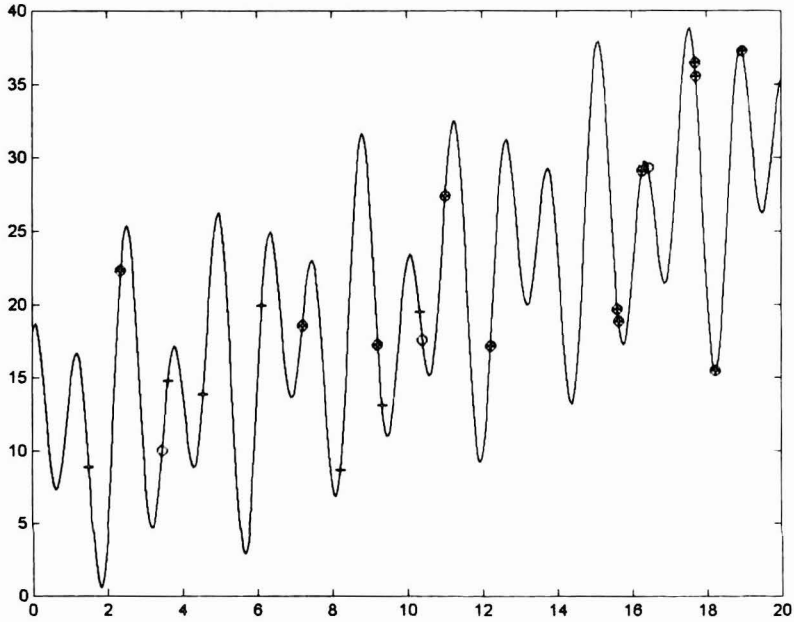


Figura 7.8: Valores obtenidos en la primera generación.

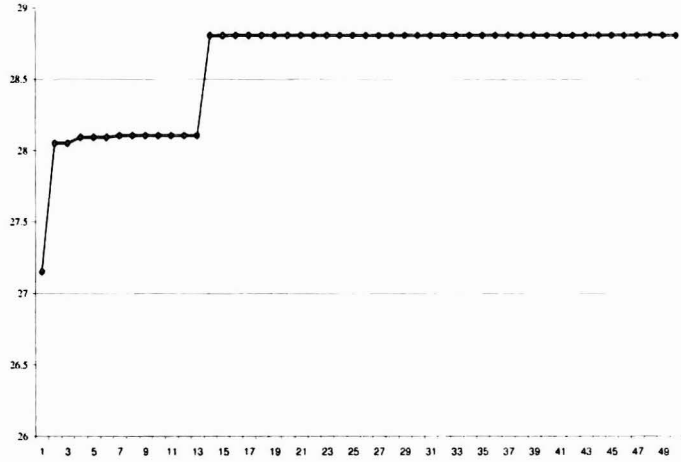


Figura 7.9: Evolución de las mejores soluciones encontradas por el algoritmo.

y para obtener una precisión de 6 dígitos después del punto decimal, habrá

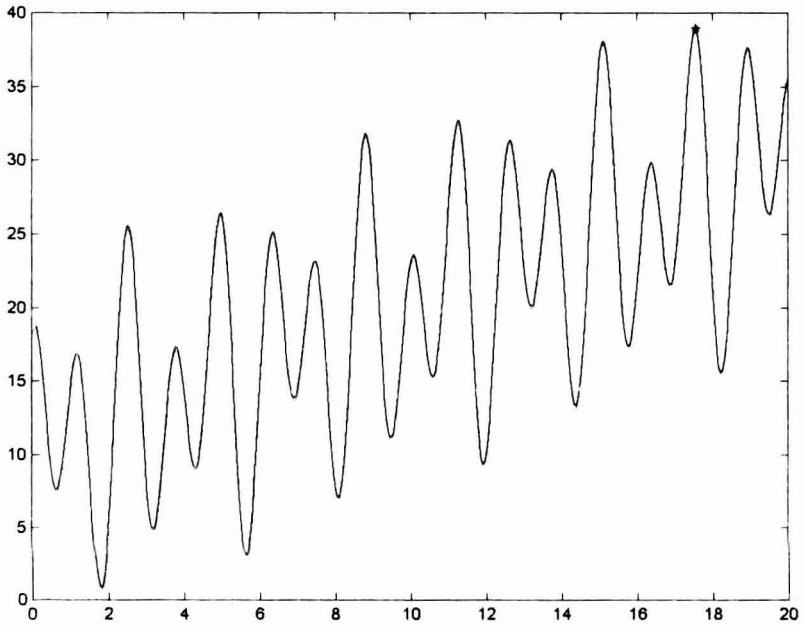


Figura 7.10: Valor máximo.

que usar 25 bits por cada variable, o sea, cromosomas con 50 bits en total.

Ahora, podemos proceder como en el Ejemplo 1, definiendo los parámetros: tamaño de la población, criterio de terminación, y las probabilidades de cruzamiento y mutación. Esta vez, tomamos el tamaño de población igual a 20, con 200 iteraciones, P_c=0.99 y P_m=0.05. En este caso tenemos que asegurar que la función no tiene valores negativos, por lo tanto usamos la función $f(x, y) + 10$ y al final restamos el 10 para obtener el verdadero valor de la función. Estos son los valores mostrados en las figuras 7.12 y 7.13.

En la figura 7.12 se muestra los mejores valores obtenido durante la corrida. El valor óptimo de la función encontrado es 28.801455 en el punto (17.5382, 17.5387).

Debido a que los resultados obtenidos dependen de los parámetros escogidos, vamos a repetir las corridas para ilustrar esta variación en los resultados. Los mejores valores obtenidos en 10 diferentes corridas fueron: 28.801455,

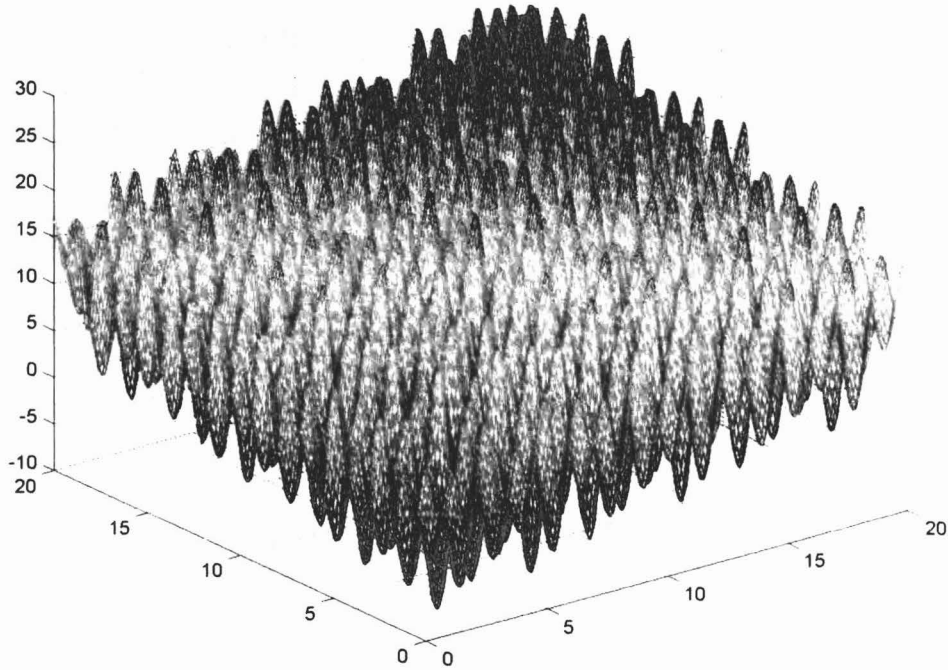


Figura 7.11: $f(x,y) = [x + 5 * sin(3 * x) + 8 * cos(5 * x) + y + 5 * sin(3 * y) + 8 * cos(5 * y)]/2$.

28.804735, 28.697641, 28.801376, 28.802042, 28.803342, 28.804734, 28.804247, 28.802126, 28.697644. Se muestran estos valores en la figura 7.13.

Una forma muy común de reportar el desempeño de un algoritmo heurístico es mediante la **mejor aptitud promedio** obtenida, que se define de la siguiente manera:

1. Medir la mejor aptitud encontrada después de un número fijo de iteraciones.

2. Tomar el promedio de los mejores valores después de muchas corridas.

La mejor aptitud promedio obtenido en las 10 corridas es 28.781934.

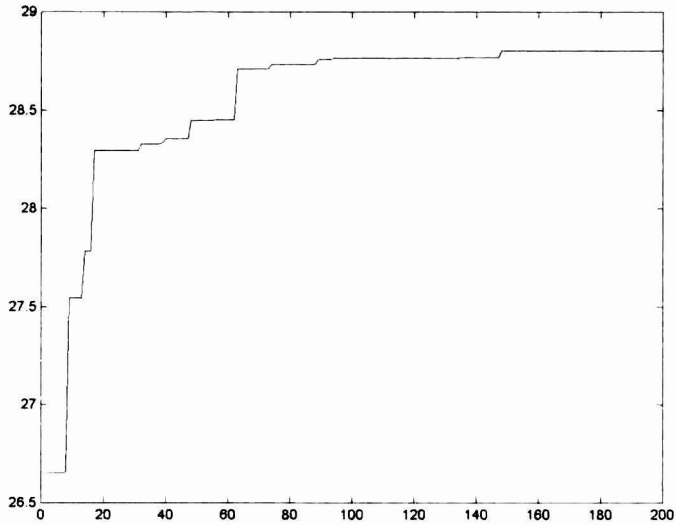


Figura 7.12: Los mejores valores obtenidos en una corrida de 200 iteraciones.

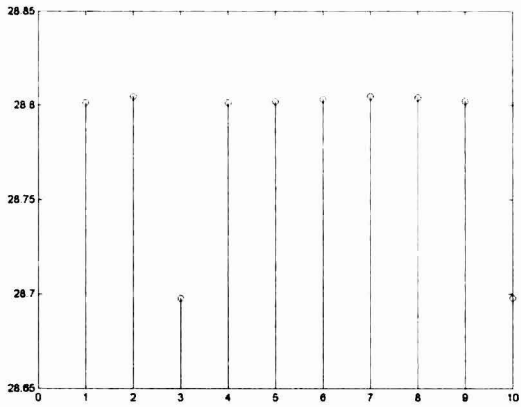


Figura 7.13: Los mejores valores obtenidos en 10 corridas del algoritmo genético.

7.3.6 Ejemplo: Agente Viajero.

Otro ejemplo típico de optimización es el que se mencionó en el capítulo 3 como NP, y es la del agente viajero. Recordando el problema: dadas n

ciudades que tiene que visitar un vendedor, encontrar el camino más corto para que, partiendo de una de ellas, visitarlas todas sin pasar más de una vez por cada una y volviendo, finalmente, a la ciudad de partida.

En una representación binaria de n ciudades, cada ciudad puede ser codificada como una cadena de $log_2 n$ bits; un cromosoma es una cadena de $n * log_2 n$ bits, es decir, cada cromosoma representa un recorrido para el agente viajero. Existe una serie de problemas con los operadores genéticos descritos en los ejemplos anteriores, para la solución de este ejemplo. En una mutación puede resultar una secuencia de ciudades la cual no es un recorrido válido ya que se puede tener la misma ciudad dos veces dentro de la misma secuencia; por otro lado, con 20 ciudades, donde se necesitan 5 bits para representar una ciudad, algunas secuencias no corresponden a alguna ciudad válida, por ejemplo 10101, ya que debido a que sólo existen 20 ciudades este número binario representaría la ciudad 21; similares problemas se presentan con el cruzamiento.

Tomando en cuenta estas dificultades, puede ser más conveniente adoptar un enfoque diferente del algoritmo genético tradicional en términos de la codificación y de los operadores empleados. Por ejemplo, en lugar de una codificación con números binarios, podemos utilizar una codificación con números enteros, que representan una enumeración de las ciudades que el agente visita. Para el caso de 8 ciudades, podemos representar un cronosoma con un ciclo de la forma (2 4 5 1 3 8 6 7), que significa que el agente visitará las 8 ciudades en el orden 2-4-5-1-3-8-6-7-2. La aptitud para este cromosoma sería la distancia total de este recorrido de ciudades.

Con esta representación tenemos que modificar los operadores genéticos de cruzamiento y mutación; se han desarrollado muchos diferentes (c.f. [30], [69]), y aquí mencionamos dos, nada más, uno para cruzamiento y uno para mutación.

(a) *Cruzamiento por ciclos* (en inglés, *cycle crossover* o CX): Escogenos dos cromosomas padres, y vamos a recombinarlos en dos cromosomas hijos.

$$p_1 = (3\ 1\ 2\ 6\ 4\ 5\ 7\ 9\ 8),\ p_2 = (4\ 5\ 2\ 1\ 8\ 7\ 6\ 9\ 3)$$

Ahora, empezando con el primer elemento de p_1, 3 en este caso, venos cual es el elemento correspondiente en la misma posición abajo en p_2, aquí

es 4. Luego, encontramos 4 en p_1 y buscamos el elemento abajo, que es 8. De esta forma obtenemos el ciclo 3-4-8-3. Pongamos estos valores en el hijo, h_1, donde x significa que todavía no sabemos el valor:

$$h_1 = (3 \text{ x x x } 4 \text{ x x x } 8)$$

Después, se llena los valores faltantes usando p_2:

$$h_1 = (3\ 5\ 2\ 1\ 4\ 7\ 6\ 9\ 8)$$

De forma similar, se obtiene h_2:

$$h_2 = (4\ 1\ 2\ 6\ 8\ 5\ 7\ 9\ 3)$$

(b) *Mutación por inversión*: se escoge una subcadena aleatoria y la invierte. Si p_1 es el cromosoma

$$p_1 = (3\ 1\ 2\ 6\ 4\ 5\ 7\ 9\ 8)$$

y se escoge la subcadena '4 5 7 9', obtenemos el nuevo cromosoma:

$$h_1 = (3\ 1\ 2\ 6\ 9\ 7\ 5\ 4\ 8)$$

Vemos que ambos operadores producen cromosomas válidos.

(c) *Método de selección por ordenamiento exponencial*: aunque podemos usar el método de la ruleta para la selección, vamos a introducir otro basado en ordenamiento exponencial [54]. La razón para utilizar otros métodos de selección es para evitar que los cromosomas con alta aptitud dominen demasiado rápido.

El nuevo método funciona dando un valor de selección a cada cromosoma basada en su orden entre la población. Se realiza de la siguiente manera:

1. Primero, se ordena la población de mejor aptitud a peor aptitud.

2. Asigna 1 al mejor cromosoma, s al segundo, s^2 al tercero, y así sucesivamente hasta el último que recibe s^{N-1}.

Esto significa que la probabilidad de escoger el r-ésimo cromosoma es:

$$\frac{1-s}{1-s^N}s^{r-1},$$

donde r es el orden del cromosoma en la población, $r = 1$ es el mejor.

En la figura 7.14, vemos que aplicando este criterio a los datos de la tabla 7.1, con $s = 0.92$, y comparando con la figura 7.1 (que fue obtenida usando las aptitudes), los 3 cromosomas con peores aptitudes aumenta su probabilidad de ser escogidos en el proceso de selección, mientras que los 3 cromosomas con mejores aptitudes tienen menor probabilidad de ser escogidos. Cambiando el valor de s se modificarán los porcentajes de los cromosomas.

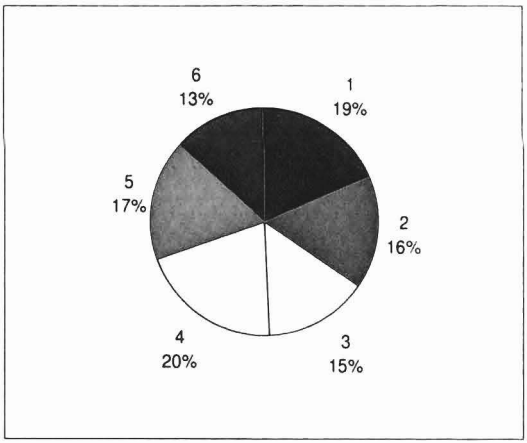


Figura 7.14: Los porcentajes asignados a los mismos seis cromosomas de la tabla 7.1, con s=0.92.

Vamos a aplicar los operadores genéticos descritos arriba en el problema del agente viajero. Para el ejemplo, tomamos el caso de 8 ciudades, que están organizadas en la forma mostrada en la figura 7.15. La distancia total para el recorrido más pequeño es de 96.5685 unidades.

Aún con el método de selección por ordenamiento exponencial, requerimos maximizar la función de aptitud, en este caso trabajamos con el valor negativo del recorrido total del cromosoma.

Para medir la evolución de la población, vamos a reportar, por cada generación, la mejor aptitud, la aptitud promedio y la desviación estándar; usamos las siguientes formulas para las últimas dos:

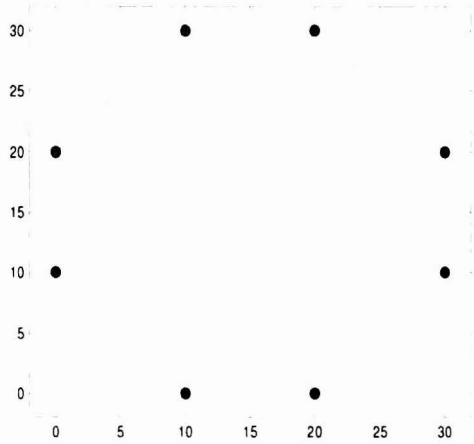


Figura 7.15: Ciudades escogidas para el ejemplo con el recorrido más corto.

$$\text{Media} \quad \bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i,$$

$$\text{Desviación estándar} \quad s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2}$$

La desviación estándar indica la variación de aptitudes en la población actual.

La figura 7.16 muestra los mejores valores y los valores promedios obtenidos en una corrida de 200 generaciones. Vemos que el mejor valor fue -96.5685 (de hecho el verdadero valor óptimo) logrado en la sexta generación, y el valor promedio coincide con el mejor en la doceava generación.

Vemos el efecto del método de selección mediante los valores de la desviación estandar en cada generación.

7.4 Estrategias Evolutivas

Las estrategias evolutivas fueron desarrollas en la década de 1960 en Alemania por Bienert, Rechenberg, y Schwefel, para atacar el problema de encontrar la forma óptima de cuerpos sujetos a viento, el cual es un problema

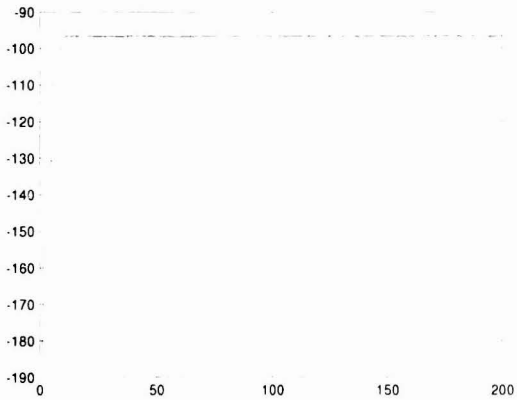


Figura 7.16: Los mejores valores y los valores promedios por cada iteración.

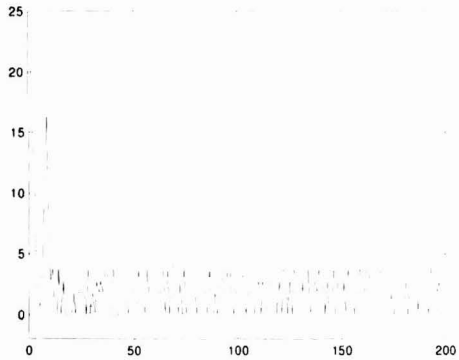


Figura 7.17: La desviación estandar en cada iteración.

de optimización numérica. Las diferencias principales entre estrategias evolutivas y algoritmos genéticos radican en que las estrategias, directamente, usan vectores de números reales en lugar de cadenas binarias, e incorporan los operadores de selección y mutación, pero no cruzamiento.

7.4.1 Algoritmos ($\mu + \lambda$)-ES y (μ,λ)-ES

Generalmente, una estrategia evolutiva empieza con $\mu > 0$ 'padres' en la población que inicialmente están escogidos aleatoriamente. Aquí la terminología usada es 'padre' o 'hijo', en lugar de 'cromosoma' para el caso de un AG. Otra vez, el padre o hijo codifica una solución posible al problema de optimización, generalmente como un vector de números reales. Se refiere al caso ($\mu + \lambda$)-ES, cuando los padres producen λ hijos, mediante un solo o-

perador genético aplicado en el proceso de evolución, denominado mutación. Luego se toma los μ mejores, según sus aptitudes, de los $(\mu + \lambda)$ padres e hijos para forma la siguiente población c.f. [52]. Realmente esta selección es una sobrevivencia de los más aptos y, automáticamente, incorpora la idea de elitismo, mencionado anteriormente. Esto se muestra en la figura 7.18. También existe una variante, llamada (μ,λ)-ES, donde se crean λ hijos con mutación, y los μ mejores de ellos son escogidos. Aquí, ningún padre sobrevive.

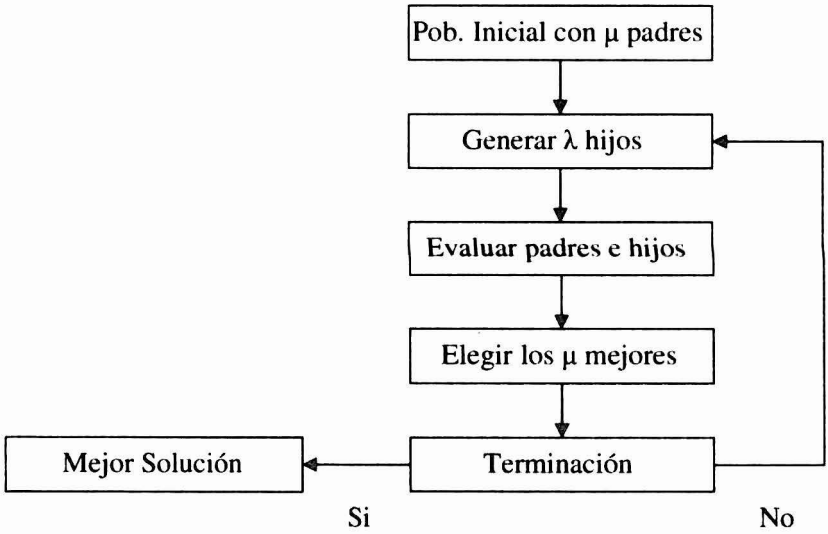


Figura 7.18: Un algoritmo $(\mu + \lambda)$-ES.

7.4.2 Algoritmos (1 + 1)-ES

Al principio, las estrategias evolutivas se basaron en (1+1)-ES, o sea, una población de un solo individuo que produce un hijo mediante el operador genético de mutación. La diferencia con los algoritmos genéticos, fue representar a un individuo como un vector de números reales, x en $\Re^n$, donde el vector x representa una posible solución en el espacio de búsqueda de la función de aptitud. La mutación se lleva a cabo remplazando a x por $x + \delta$, donde δ es un vector de números aleatorios de la distribución normal o gaussiana, $N(0, \sigma_i^2)$, con media igual a cero y desviaciones estándares σ_i, $i = 1, \ldots, n$. Esta convención se dió debido a que en la evolución natural

Los parámetros de entrada son: el número de padres, μ, el número de hijos, λ, tipo de mutación y la condición para la terminación.

Comienza

inicializa la población

repetir

comienza

aplicar mutación a ciertos padres de la población

evalúa los padres e hijos

elegir los μ mejores

termina

hasta condición de terminación

devuelve la solución más apta en la población

termina

Figura 7.19: Pseudocódigo para un algoritmo $(\mu + \lambda)$-ES.

ocurren más a menudo pequeños cambios en lugar de cambios drásticos. Se acepta el resultado de la mutación como un nuevo miembro de la población remplazando a su padre siempre y cuando tenga mejor aptitud que éste, es decir, sea f la función objetivo, el hijo, $x + \delta$, reemplaza a su padre x si $f(x + \delta) > f(x)$. En otro caso el hijo se elimina y la población se mantiene sin cambios.

7.4.3 Ejemplo: Optimización de funciones reales con un algoritmo (1+1)-ES

Otra vez buscamos el valor máximo de la función:

$$f(x) = x + 5 * sin(3 * x) + 8 * cos(5 * x)$$

en el intervalo [0,20], pero esta vez usando una estrategia evolutiva. Por el funcionamiento del algoritmo (1+1)-ES, no es necesario que la función sea positiva, como en el ejemplo 7.3.4.

Inicialmente, escogemos un número aleatorio en el intervalo [0,20] para ser el primer padre. Luego, cuando aplicamos el operador de mutación gaussiana, tenemos cuidado en no permitir que el hijo salga del intervalo [0,20]. de las soluciones factibles. Si en algún momento el nuevo valor rebasa el límite inferior o superior del intervalo, entonces asignamos el valor de 0 o 20, respectivamente.

Vamos a tomar el criterio de terminación de 500 iteraciones, y la desviación estándar de la distribución como $\sigma = 0.01$, 0.05, 0.1, 0.5, 1.0, 1.5, 2.0, 3.0 de manera sucesiva en corridas independientes.

Los mejores valores encontrados para los diferentes valores de σ se muestran en la tabla 7.3. Habíamos visto que el mejor valor encontrado anteriormente era 28.8023 en el punto 17.5393. En el caso actual, vemos que los valores de σ más grandes son los que dan los mejores resultados. Podemos ver, que cuando el valor de la función es en un máximo local, digamos x, se requiere una perturbación δ de la distribución gaussiana, $N(0, \sigma^2)$, lo suficientemente grande para que el hijo, $x + \delta$, puede alejarse y obtener un aptitud mayor; de otra forma, se queda atrapado en el máximo local. Por ejemplo, si el valor del padre fuera 0, entonces para salir de este máximo local obteniendo un hijo con mayor aptitud, se requiere una perturbación con un valor mayor a 2 (c.f. figura 7.20). Ahora se sabe que para obtener un valor positivo aleatorio de la distribución normal que cae a más de tres desviaciones estándares de la media (aquí 0), requiere una probabilidad menor de 0.002, un valor 'poco probable' porque ocurre aproximadamente 2 veces en 1000 intentos. Esto significa que si $\sigma = 0.1$, para tres desviaciones estándares, $3\sigma = 0.3$ es muy poco probable que una perturbación puede lograr escapar del máximo local de 0.

A pesar de las últimas observaciones del ejemplo, es posible mostrar que para funciones continuas y definidas en intervalos cerrados con la desviación estándar fija, se encontrará el óptimo global con probabilidad de 1 para tiempos de búsqueda suficientemente largos c.f. [12]. Sin embargo, de no proveer ninguna pista sobre, por ejemplo, el número de generaciones necesarias, se limita su relevancia en la practica. En [43] se encuentran algunas respuestas a este tema.

Desviación Estándar σ	Mejor punto	Mejor aptitud
0.01	8.8197	21.6171
0.05	18.9294	27.4866
0.1	16.3729	19.6820
0.5	17.5420	28.8044
1.0	17.5439	28.8047
1.5	17.5868	28.5957
2.0	17.5300	28.7831
3.0	17.5573	28.7843

Tabla 7.3: Los mejores aptitudes encontrados con los diferentes valores de σ.

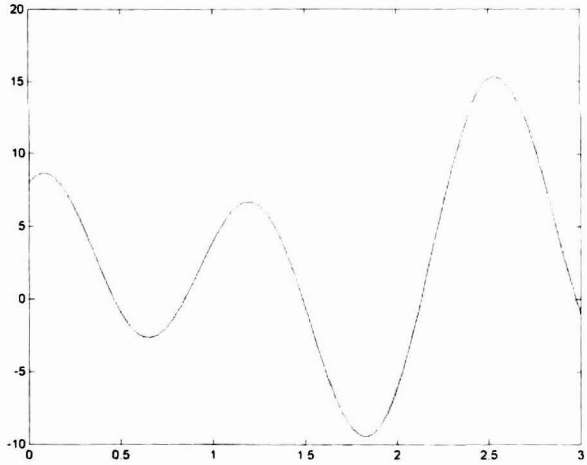


Figura 7.20: Gráfica de la función en el intervalo [0,3].

7.4.4 Adaptación determinista de los parámetros

Cuando se permite que la desviación estándar cambia periódicamente en el algoritmo, es conocido como auto-adaptación. Rechenberg propuso la siguiente regla heurística para modificar el valor de la desviación estándar, σ, conocida como 'la regla de éxito 1/5', durante los algoritmos (1+1)-ES: encuentra el porcentaje, p_s, de mutaciones que tuvieron mayores aptitudes que sus padres (consideradas como exitosas) en alrededor de $10n$ experimentos, donde n es la dimensión de x. Entonces, con $c = 0.85$, se modifica σ de la

siguiente forma:

$$\sigma = \sigma/c \text{ si } p_s > 1/5,$$

$$\sigma = c\sigma \text{ si } p_s < 1/5,$$

$$\sigma = \sigma \text{ si } p_s = 1/5.$$

La auto-adaptación puede mejorar el desempeño de un algoritmo, pero no garantiza su convergencia a un óptimo global, y de hecho, por razones similares a los citados en el ejemplo 7.4.3, se puede permanecer en un máximo local. Greenwood [52] modificó la regla de Rechenberg de la siguiente manera:

$$\sigma = \min(\sigma/c, D) \text{ si } p_s > 1/5,$$

$$\sigma = c\sigma \text{ si } 1/20 \leq p_s < 1/5,$$

$$\min(2\sigma, D) \text{ si } p_s < 1/20,$$

$$\sigma{=}\sigma \text{ si } p_s = 1/5,$$

donde D es el diámetro del espacio de búsqueda. La modificación principal es cuando $p_s < 1/20$, en este caso se aumenta σ, esto ayuda a escapar de máximos locales. Con esta regla, [52] demostró un teorema de convergencia al óptimo global, con probabilidad uno, para una clase de funciones objetivas generales.

7.4.5 Ejemplo: Optimización de funciones reales con un algoritmo (1+1)-ES auto-adaptivo.

Repetimos el ejemplo incorporando la regla de [52], con $D = 20$. Los demás parámetros permanecen iguales.

Vemos los resultados de una corrida en la tabla 7.4. En este caso, aún para valores pequeños de σ, con la auto-adaptación, se ha podido escapar de los máximos óptimos.

En la figura 7.21, se ve la evolución de la desviación estándar cuando se inició con el valor de 0.01.

Desviación Estándar σ	Mejor punto	Mejor aptitud
0.01	17.5174	28.7263
0.05	17.4806	28.3633
0.1	17.5256	28.7671
0.5	17.5370	28.7994
1.0	17.5745	28.6986
1.5	17.6032	28.4056
2.0	17.5894	28.5692
3.0	17.6269	28.0229

Tabla 7.4: Los mejores aptitudes encontrados con los diferentes valores de σ.

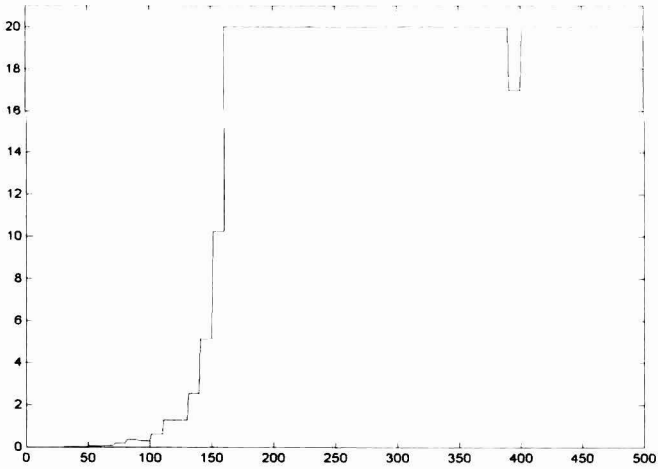


Figura 7.21: Evolución de la desviación estándar iniciando con $\sigma = 0.01$.

7.4.6 Adaptación aleatoria de los parámetros de la mutación

En lugar de utilizar una regla determinista para la auto-adaptación de los parámetros de mutación, se puede considerar a los parámetros como parte de la representación y también codificarlos c.f. [44]. En este caso, la representación tiene la forma $(x_1, \ldots, x_n, \sigma_1, \ldots, \sigma_n)$, donde $(x_1, \ldots, x_n)$ es un punto en el espacio de búsqueda, y $(\sigma_1, \ldots, \sigma_n)$ son las desviaciones estándares iniciales correspondientes a las mutaciones.

En el caso más sencillo, donde cada parámetro de mutación tiene el mismo valor, σ, se obtiene el hijo, $(x_1^{'}, \ldots, x_n^{'}, \sigma^{'})$, del padre $(x_1, \ldots, x_n, \sigma)$ de la siguiente forma:

1. $\sigma^{'} = \sigma e^{t\delta}$, donde δ es un número aleatorio de la distribución normal N(0,1).
2. $x_i^{'} = x_i + \delta^{'}$, donde $\delta^{'}$ es un número aleatorio de la distribución normal N(0,$\sigma^{'}$).

donde t es una constante fijada por el usuario y, usualmente, proporcional a $\frac{1}{\sqrt{n}}$. Para el caso de desviaciones estándares diferentes, el lector puede consultar [44].

7.4.7 Múltiples padres

En 7.4.1 se describió los algoritmos (μ + λ)-ES y (μ,λ)-ES, donde el número de padres e hijos puede ser mayor a 1. En la práctica, es común tomar el valor de λ más grande que μ (frecuentemente $\lambda = 7\mu$). Recordamos que en el caso (μ + λ)-ES, se generan λ hijos a partir de los μ padres, utilizando un operador de mutación, y después se toma los μ mejores de todos para la siguiente generación. Para el algoritmo (μ,λ)-ES, se toma los μ mejores de los λ hijos generados nada más, y los padres nunca sobreviven.

¿ Cuáles padres usamos para genera los hijos? No se toma los padres más aptos, como es usual para algoritmos genéticos, sino se escogen de manera aleatoria de la población de individuos.

Además, con múltiples padres podemos introducir un nuevo operador llamado recombinación ó cruzamiento. Dos maneras comunes de hacerlo son:

Sean $(x_1, \ldots, x_n)$, $(y_1, \ldots, y_n)$ dos individuos y formar un nuevo individuo $(z_1, \ldots, z_n)$ según:

1. recombinación discreta: cada z_i es x_i o y_i.
2. recombinación intermedia: $z_i = \frac{(xi+yi)}{2}$.

Nota: como hemos visto en 7.4.6, un individuo puede incluir parámetros también.

7.5 Programa Evolutivo de Cromosomas de Longitud Variable

En los métodos discutidos hasta el momento, se ha visto que éstos involucran una población que consiste en una colección de posibles soluciones al problema bajo consideración, llamados cromosomas o padres. Más aún, se ha supuesto que cada una de las soluciones tiene una codificación como una cadena binaria o un vector, todos con los mismos tamaños o dimensiones. Sin embargo, hay problemas para los cuales las posibles soluciones pueden tener una codificación donde es más natural que los tamaños pueden variar. Este tipo de formulación se conoce como *cromosomas de longitud variable*. En la presente sección, vamos a considerar un problema de aprendizaje en los perceptrones multicapa donde es conveniente adoptar una codificación con cromosomas de longitud variable.

7.5.1 Aprendizaje en los perceptrones multicapa (PMC)

En el capítulo 3, se introdujo los PMC y se consideró problemas de aprendizaje como los siguientes: Dado un conjunto de vectores $X = \{(x_1, d_1), \ldots, (x_N, d_N)\}$ con x_i en $\mathbb{R}^n$ y d_i en $\mathbb{R}^m$, para $i = 1, \ldots, N$, ¿existen pesos y umbrales tal que $F(x_i) = d_i$ para cada $i = 1, \ldots, N$, donde F es la función definida por el PMC?

Aquí, vamos a considerar el problema de clasificación, donde los vectores en X están previamente clasificados en una de un número finito de clases. Normalmente las salidas, d_i, y la dimensión, m, están codificadas de acuerdo a esas clases. Esto quiere decir que si existen dos clases se puede trabajar con un nodo de salida y valores de $\{0, 1\}$, donde la salida de la red determina si pertenece o no a una clase. En el caso de que sean más clases, cada nodo de salida represente una clase y al final se "encienda" el nodo que corresponda a la clase que el PMC encontró como solución al conjunto de valores de atributos que se le proporcionó.

El problema de clasificación también involucra la determinación de la arquitectura del PMC, o sea:

1. Número de nodos de entrada así como los de salida.

2. Número de capas ocultas.

3. Número de nodos ocultos para cada capa oculta.

Para la solución del problema de aprendizaje, algunos de estos aspectos se definen automáticamente. Normalmente el número de nodos de entrada queda determinado por el número de atributos que aparecen en el problema a resolver, y el número de nodos de salida lo define el número de atributos que aparecen en las salidas (en el problema mencionado arriba, n y m, respectivamente).

Con respecto al número de capas ocultas, se ha demostrado que los PMC con una capa oculta tienen la capacidad de aproximar uniformemente, con cualquier grado de precisión, funciones continuas multivariadas [55], aunque sin decir cuantos nodos ocultos requieren. Por lo tanto, con este resultado consideremos PMC con una capa oculta.

Queda entonces la determinación de los nodos ocultos. Como no hay un método para escoger los nodos ocultos, una forma común de proceder es mediante un proceso tardado de acierto y error, intentar con diferentes valores y ver cuál PMC da mejores resultados, después de encontrar sus pesos en cada caso. Para encontrar los pesos numéricos habitualmente se hace uso del algoritmo de retropropagación. Este algoritmo se basa en el método de gradiente descendente y requiere una función de transferencia diferenciable, así como de fijar ciertos parámetros como la tasa de aprendizaje.

Esta forma de determinar el número de nodos ocultos es muy engorrosa, se podría pensar en escoger un número *grande* de nodos ocultos; sin embargo, esto ocasionaría un problema importante para el caso de clasificación que vamos a considerar. El propósito del PMC es fungir como un clasificador para predecir si un ejemplo pertenece o no a una clase. Lo importante, entonces, es lo que se llama "su poder de generalización" sobre los casos nuevos, es decir, qué tan bien clasifica sobre casos no vistos previamente. Cuando el número de nodos ocultos es demasiado grande, puede ocurrir un fenómeno donde el PMC clasifica muy bien sobre los ejemplos conocidos que se usaron para determinar los pesos, pero no sobre nuevos ejemplos. Lo idóneo sería adoptar otro enfoque en la determinación de un PMC que podría especificar todos los parámetros involucrados simultáneamente, tanto la arquitectura como los pesos numéricos. Es aquí donde entra la computación evolutiva para la solución a este problema.

Para resolver el problema de la determinación de la arquitectura, y encontrar los pesos y umbrales requeridos de un PMC, se han realizado varios

trabajos de investigación dedicados a aplicar ideas evolutivas que han producido buenos resultados (c.f. [84], [9], [50], [112], [113], [77], [78], y [114]). No todos usan cromosomas de longitud variable, y el tema sigue siendo vigente en la investigación.

A continuación se describe un programa evolutivo con cromosomas de longitud variable, y su aplicación a la determinación de los pesos y la arquitectura de un PMC para la solución de problemas de clasificación (c.f. [77], [78]).

7.5.2 Descripción del programa evolutivo de cromosomas de longitud variable

Para aplicar un programa evolutivo en un caso concreto, primero hay que decidir cómo se va a modelar el problema; esto implica la codificación de las soluciones potenciales del problema. En nuestro caso, la idea del programa evolutivo es definir una población de cromosomas en donde cada cromosoma represente un PMC. Consideremos al PMC con una sola capa oculta donde el número de nodos ocultos pueden variar. El número de nodos de entrada y de salida son constantes, dependiendo del problema considerado. Un cromosoma deberá codificar el número de nodos ocultos que tendrá esa capa, y los pesos que unen a los nodos de entrada con los de la capa oculta, y los de ésta hacia los nodos de salida, así como los pesos de los umbrales, o sesgos, que se definieron en la capa de entrada y en la capa oculta. Esto nos lleva a la necesidad de considerar cromosomas con una longitud variable. Cada cromosoma, o PMC, está implementado como una lista donde cada elemento de la lista representa conceptualmente un nodo oculto del PMC.

La información que se guarda en un elemento de la lista son los valores de los pesos numéricos sobre las conexiones que llegan al nodo oculto de los nodos de entrada, y los que salen de los nodos ocultos hacia los nodos de salida. Un cromosoma típico se muestra en la figura 7.22 junto con el PMC que representa.

La población está guardada en un arreglo de estructuras de tamaño n que define $n-1$ cromosomas. La estructura mantiene información sobre la aptitud de cada cromosoma, lo cual se explicará más adelante, además de un apuntador a la lista del cromosoma. La posición n del arreglo se utiliza para dejar el mejor PMC encontrado de una generación a otra. Véase figura 7.23.

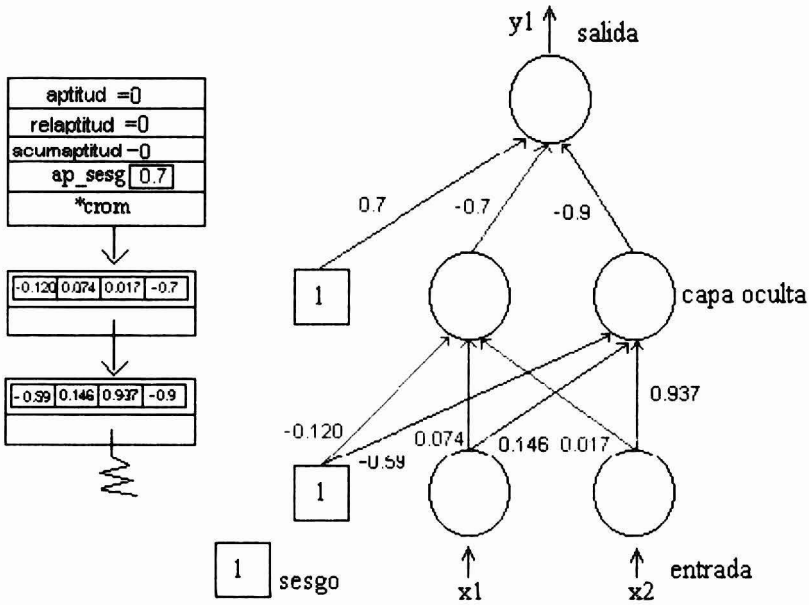


Figura 7.22: Representación de un PMC mediante un cromosoma en el programa evolutivo.

El Algoritmo Principal

El algoritmo comienza con la inicialización de una población de cromosomas y la evaluación de cada uno de sus elementos. El tamaño de la población es fijo, aunque la longitud de cada cromosoma puede variar. La población evoluciona varias generaciones aplicando los operadores de selección para escoger la nueva población, la mutación estructural para modificar el número de nodos en un cromosoma, y la mutación de los pesos.

A continuación se describen los operadores con mayor detalle.

Inicialización de la Población

Una función inicializa cada elemento de la población de la siguiente manera: el número de nodos ocultos representados en cada cromosoma se generan de manera aleatoria, cuidando que no exceda un número máximo definido por el usuario. De igual manera los pesos de cada nodo se inicializan de forma aleatoria con valores entre -1 y 1.

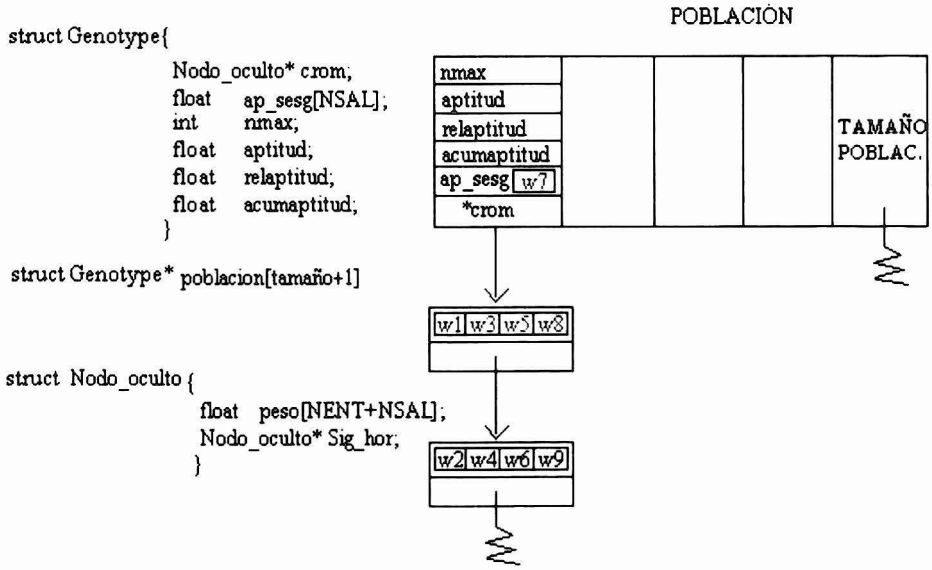


Figura 7.23: Estructuras de datos del programa evolutivo.

Función de Aptitud

La función de aptitud debe indicar la aptitud de cada cromosoma, que en nuestro contexto significa qué tan bien el PMC, correspondiente al cromosoma, clasifica un conjunto de datos. Consideremos dos diferentes maneras de calcular ésta función, aunque hay otras posibilidades. Para explicarlas hay que definir la manera exacta en la cual un PMC calcula su salida. En el caso del PMC de la figura 7.22, la entrada a cada uno de los nodos ocultos es el producto interno del vector de los pesos entrando en el nodo con el vector $(1, x_1, x_2)$, dando:

$$\text{-}0.12 + 0.074 * x_1 + 0.017 * x_2,$$

$$\text{-}0.59 + 0.146 * x_1 + 0.937 * x_2.$$

La función de transferencia que se ocupa es la función sigmoide definida de la siguiente manera: $f(u) = \frac{1}{(1+\exp^{-u})}$. Se aplica esta función de transferencia para calcular la salida de cada uno de los nodos ocultos dando:

$$z_1 = f(\text{-}0.12 + 0.074 * x_1 + 0.017 * x_2),$$

$$z_2 = f(\text{-}0.59 + 0.146 * x_1 + 0.937 * x_2)$$

Finalmente, para calcular la salida del nodo en la capa de salida, se vuelve a aplicar la función de transferencia al producto interno del vector de los pesos que entran en el nodo con el vector $(1, z_1, z_2)$, dando:

$$y_1 = f((0.7 - 0.7 * z_1 - 0.9 * z_2).$$

La función de aptitud se basa en el error cuadrático medio, ECM, que comúnmente se usa en el algoritmo de retropropagación para determinar el error. El ECM se define de la siguiente manera:

$$ECM = \frac{1}{N}\sum_{i=1}^{N}(y_i - d_i)^2$$

donde la suma es sobre todos los patrones de los datos, y_i es la salida del PMC para el patrón i, calculado como se describió anteriormente, y d_i es la salida deseada, que sería el valor que queremos asociar con el patrón. Como se quiere minimizar la función ECM, en el programa evolutivo se modifica la función para que tenga valores positivos y sea un problema de maximización.

Es interesante notar que el enfoque de programas evolutivos permite el uso de funciones de aptitud y transferencia que no son necesariamente continuas. Por ejemplo, otra función muy importante para problemas de clasificación calcula cuántos errores tiene el PMC al clasificar, es decir, cuántos datos están mal clasificados. Esta es una ventaja del programa evolutivo sobre el algoritmo de retropropagación para un PMC.

El campo de aptitud en la estructura de datos corresponde al valor de la función de aptitud.

Selección de la Nueva Población

Para escoger una nueva población se aplica el método de la rueda de la ruleta (c.f. 7.3.1); sin embargo, es importante mencionar que no es el único método que existe. La aptitud relativa y la acumulada son campos de la estructura que se guardan en el arreglo (figura 7.23).

Mutación Estructural

Es necesario tener al menos un operador que pueda cambiar el tamaño de los cromosomas; si no fuera así, el número de nodos ocultos en cada cromosoma quedaría igual al número con el cual fue inicializado. Esto es importante de acuerdo a nuestra meta de encontrar simultáneamente la arquitectura y los pesos más adecuados de un PMC para resolver un problema de clasificación. El programa evolutivo utiliza dos operadores, Adicionar y Eliminar, definidos como sigue:

1. Adicionar, se encarga de aumentar un nodo oculto al cromosoma en cuestión inicializando sus pesos con 0. Se valida que al aumentar este nodo no sobrepase el número máximo de nodos permitidos por el usuario; si ya se tiene ese número máximo no se añade otro nodo. Este nodo se incorpora al final de la lista y se hace de acuerdo a un número generado de forma aleatoria: si éste es menor a una probabilidad de adición definida por el usuario se adiciona el nodo, en otro caso no se hace ningún cambio al cromosoma.

2. Eliminar, para equilibrar el efecto que pudiera tener la función Adicionar, y para reducir el número de nodos ocultos en un cromosoma, se define otro operador de nombre Eliminar. De forma similar se genera un número aleatorio y se compara con la probabilidad de eliminación definida por el usuario, si dicho número es menor se elimina el primer nodo de la lista del cromosoma en cuestión.

En los dos casos se reduce linealmente el parámetro de probabilidad de su valor inicial a un valor pequeño definido por el usuario, conforme evoluciona el algoritmo; con esto, al final de la corrida no es necesario modificar la arquitectura del PMC.

Mutación Paramétrica

Para modificar los pesos de un cromosoma, se aplica un operador a los pesos de un nodo oculto del cromosoma. Esto quiere decir que para cada nodo oculto de un cromosoma se genera un número aleatorio y si éste es menor a la probabilidad de mutación definida por el usuario, se modifican los pesos agrupados en el nodo; en caso contrario se conserva el mismo valor de los pesos. Si se decide cambiar el valor de un peso se le agrega el valor generado por

una distribución gaussiana, con media 0 y varianza definida por el usuario. La varianza está reducida linealmente conforme evoluciona el algoritmo; esto se hace para evitar perturbaciones *grandes* a los pesos lo cual no es benéfico al final de la corrida.

Guardar el mejor cromosoma

El cometido de este elitismo es asegurar que el mejor cromosoma encontrado sobreviva de una generación a otra. En el programa evolutivo, si el cromosoma con mejor aptitud de la población actual supera al seleccionado en la población anterior, éste es reemplazado, conservando siempre al mejor cromosoma de una generación a otra. En otro caso reemplaza el peor cromosoma de la población actual con el mejor de la generación anterior.

El algoritmo principal del programa evolutivo se muestra en el cuadro 7.24.

Ahora, vamos a aplicar el programa evolutivo descrito en 7.5.2 a varios problemas de clasificación.

7.5.3 Ejemplo: Clasificación mediante un PMC definido por un programa evolutivo

A continuación, se describen seis bases de datos con las cuales se probó el desempeño del programa evolutivo propuesto, después de su descripción se muestran los resultados obtenidos en cada una de estas bases de datos.

Descripción de las bases de datos

Las bases de datos tienen diferentes grados de complejidad en la clasificación de sus datos además de tener diferente número de ejemplos, de atributos y clases.

1. Base de datos IRIS

   Base de datos creada por R. A. Fisher y donada por Michael Marshall en julio de 1988 [109]. Esta base de datos es quizá, la más conocida en la literatura de reconocimiento de patrones. Consta de 150 ejemplos, 3 clases y 4 atributos, la distribución de los ejemplos es 50 ejemplos

```
Comienza
    Generación=0
    Inicializar la población
    Evaluar cada cromosoma
    Guardar el mejor cromosoma
    Mientras (generación < MAXGENS)
    Comienza
        Seleccionar la nueva población
        Aplicar Mutación Estructural
        Aplicar Mutación de los pesos
        Evaluar cada cromosoma
        Guardar el mejor cromosoma
        Generación=Generación+1
    termina
termina
```

Figura 7.24: Algoritmo Principal del Programa Evolutivo.

para cada clase. No se cuentan con datos faltantes. Cada clase se refiere a un tipo de planta iris, las cuales son las siguientes: iris setosa, iris versicolor e iris virginica. Una clase (iris setosa) es linealmente separable de las otras dos, (es decir, separable mediante un hiperplano), lo cual se puede observar en la figura 7.25; sin embargo, la clase iris versicolor e iris virginica cuentan con ejemplos traslapados evitando ser linealmente separables entre sí. Los atributos son el largo y ancho del sépalo y el largo y ancho del pétalo.

En [31] se reporta el 100% de clasificación de la clase iris setosa y muy pocos ejemplos mal clasificados para el resto de las clases.

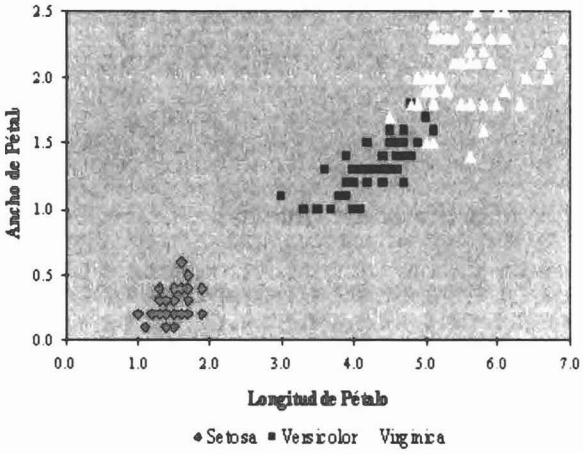


Figura 7.25: Gráfica de las clases del ejemplo de IRIS.

2. Base de datos de los indios PIMA

   Base de datos original del Instituto Nacional de Diabetes y Enfermedades del Riñón y Digestivas su donante: Vincent Sigillito, Applied Physics Laboratory, The Johns Hopkins University, su fecha de recepción es del 9 de mayo de 1990, [109].

   En esta base de datos se determinaron algunas restricciones las cuales se mencionan a continuación; de la base de datos original se seleccionaron todos los pacientes del sexo femenino de al menos 21 años de edad y descendentes de la tribu de indios Pima. Cuenta con 768 ejemplos y aunque no se reportan datos faltantes, al examinar la base de datos se encuentran una gran cantidad de ejemplos con atributos sin valor, reduciendo la base de datos a 336 ejemplos, cuya distribución es la siguiente: 225 ejemplos corresponden a la clase 1 (prueba de diabetes positiva) y 111 a la clase 0 (prueba de diabetes negativa). El número de atributos es de 8 y cuenta con dos clases, como ya se había mencionado.

   Los atributos son los siguientes:

   (a) Número de embarazos.

   (b) Concentración de glucosa en plasma a dos horas en una prueba de tolerancia oral a la glucosa.

(c) Presión sanguínea diastólica (mm Hg).

(d) Medición del pliegue en el tríceps (mm).

(e) Insulina en Serum 2 horas (mu U/ml).

(f) Índice de masa corpórea (peso en Kg/(estatura en m)^2).

(g) Función de pedigree de diabetes.

(h) Edad (años).

En [99] reportan, utilizando un algoritmo de aprendizaje denominado ADAP, el 76% de aciertos sobre los datos de prueba.

3. Desordenes Cardiacos

Esta base de datos es compilada originalmente en la Cleveland Clinic Foundation y suministrada por Robert Detran M.D., Ph. D. Del V. A. Medical Center, Long Beach CA, es parte de una colección de datos de la Universidad de California recolectada por David Aha [109].

El propósito de esta base de datos es predecir la presencia o ausencia de enfermedad cardiaca dados los resultados de varias pruebas médicas llevadas a cabo sobre el paciente. La base de datos posee 13 atributos que fueron extraídos de un conjunto mayor de 75. Los 13 atributos son los siguientes:

1: Edad.

2: Sexo.

3: Dolor de pecho (4 valores).

4: Presión sanguínea en reposos.

5: Colesterol (mg/dl).

6: Nivel de azúcar en sangre > 120mg/dl (en ayunas).

7: Resultados electrocardiográficos en reposo (valores 0,1,2).

8: Máxima frecuencia cardiaca alcanzada.

9: Angina inducida por ejercicio.

10: Depresión del segmento ST inducida por ejercicio con respecto al reposo.

11: Pendiente del segmento ST (pico del ejercicio).

12: Número de vasos mayores coloreados por fluoroscopia (0-3).

13: Thal.: 3=normal, 6=defecto fijo, 7=defecto reversible.

El conjunto de datos original contenía 303 ejemplos, pero algunos fueron descartados quedando tan solo 270, y existen dos clases: presencia o ausencia de enfermedad cardiaca. La distribución de ejemplos de cada clase es la siguiente: 150 ejemplos son sanos lo cual corresponde al 55.56% del total y 120 ejemplos corresponden a la clase de enfermos lo cual equivale a un 44.44% del total.

Los mejores resultados publicados en [50] sobre esta base de datos es del 81% de aciertos sobre los datos de prueba utilizando un algoritmo hibrido (algoritmo genético y retropropagación), y del 77.5% utilizando únicamente un algoritmo genético.

4. Base de datos Peterson Barney

Peterson Barney es una base de datos que reporta las cuatro primeras formantes generadas por la señal de la voz de 33 hombres, 28 mujeres y 15 niños, teniendo un total de 76 personas produciendo dos repeticiones de 10 vocales, obteniendo así 1520 muestras de voz en el idioma inglés [46].

Las vocales generadas son las siguientes:

1 IY [i]

2 IH [I]

3 EH [e]

4 AE [ae]

5 AH [ˆ]

6 AA [a].

7 AO [o]

8 UH [U]

9 UW [u]

10 ER [3]

Existen ejemplos en la base de datos donde los expertos no logran identificar a que vocal pertenecen por lo tanto estos han sido eliminados. Para probar el algoritmo propuesto se escogieron los ejemplos de las dos vocales más difíciles de clasificar: UH y UW debido al gran traslape de sus datos. En la Figura 7.26 se muestra su distribución. y se decidió eliminar el primer atributo ya que se considera F1, F2 y F3 las formantes con mayor información.

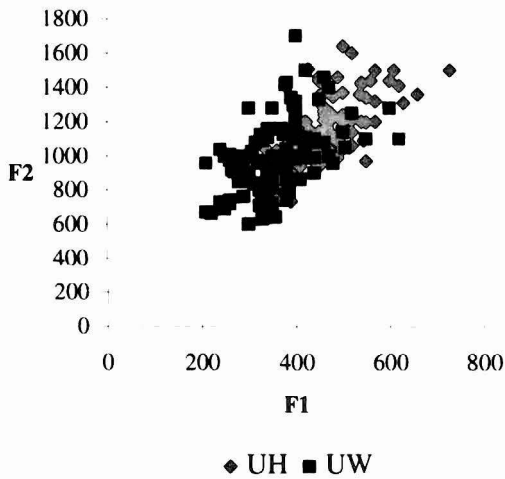


Figura 7.26: Ejemplo de la base de datos Peterson Barney.

Del total de ejemplos de la base de datos original se tomaron 275, de los cuales 132 corresponden a la clase UH y 143 a la clase UW.

5. Localización de proteínas

Esta base de datos es creada por Kenta Nakai del Institue of Molecular and Cellular Biology Osaka, University y donada por Paul Horton en 1996 [109]. Consta de 336 ejemplos y siete atributos, los cuales se mencionan a continuación:

1. mcg: método McGeoch's para el reconocimiento de la señal de secuencia.

2. gvh: método von Heijne's para el reconocimiento de la señal de secuencia.

3. lip: consenso para la secuencia de la señal von Heijne's Signal Peptidase II.

4. chg: Presencia de carga para predicción de lipoproteinas.

5. aac: Puntuación del análisis discriminante del contenido aminoacido de la membrana externa y proteinas periplamica.

6. alm1: puntuación del programa ALOM.

7. alm2: puntuación del programa ALOM después de excluir la región para la señal de la secuencia.

No existen valores faltantes. Las clases son el sitio de localización de proteínas,y son ocho que a continuación se mencionan con la distribución de los ejemplos:

cp (citoplasma) 143,

im (membrana interior sin señal de secuencia) 77,

pp (perisplasma) 52,

imU (menbrana interna, sin hendidura en la señal de secuencia) 35,

om (membrana externa) 20,

omL (membrana externa lipoproteina) 5,

imL (membrana interna lipoproteina) 2,

imS (membrana interna, con hendidura en la señal de secuencia) 2.

En [59] se reporta un 81% de aciertos con un modelo probabilístico y un porcentaje similar con un árbol de decisión binario y un clasificador Bayesiano

6. Reconocimiento de vino

Creada por el Instituto de Farmacéutica y Análisis y Tecnología de la Comida en Italia y dada de alta en 1998 por C. Blake [13]. Estos datos son los resultados de un análisis químico de vinos de la misma región de Italia, pero derivado de diferentes cultivos de árboles. El análisis determina 13 componentes encontrados en cada uno de los tipos de árboles de vinos.

Los atributos son estos 13 componentes y se enlistan a continuación:

1) Alcohol,

2) Malic acid,

3) ash,

4) Alcalinity of ash,

5) Magnesio,

6) Total phenols,

7) Flavanoids,

8) Nonflavanoid phenols,

9) Proanthocyanins,

10) Intensidad del color,

11) Matiz,

12) OD280/OD315 de vinos diluidos,

13) Proline.

Existen 3 clases cuyo nombre no es reportado y su distribución es la siguiente:

Clase 1 59 ejemplos,

Clase 2 71 ejemplos,

Clase 3 48 ejemplos.

En total 178 ejemplos con ningún valor faltante.

Aplicando varios algoritmos para su clasificación se reportan los siguientes resultados [3]:

Algoritmo	Porcentaje de aciertos
RDA	100%
QDA	99.4%
LDA	98.9%
INN	96.1%

Resultados

Para aplicar el programa evolutivo a cada uno de las seis bases de datos, separamos cada base en dos conjuntos llamados entrenamiento, E, y prueba, P. Se aplica el programa evolutivo de 7.5.2 al conjunto E para encontrar la arquitectura y los pesos y umbrales del PMC. A este PMC se llama el PMC *entrenado*. Después, se calcula las salidas de cada patrón en P, utilizando el PMC entrenado, y se compara estas salidas con su verdadera clase para encontrar el número de patrones correctamente clasificados. Luego se calcula la tasa de aciertos para P usando la siguiente formula:

$$\text{Tasa de aciertos} = \frac{\text{número correctos}}{\text{número total}} \times 100$$

En las tablas 7.5 y 7.6 se presentan las características más relevantes de cada base de datos.

Nombre de la BD	Clasific.	Atrib.	Clases	Núm. total	Entren.	Prueba	% aciert. reportado
Iris	Plantas	4	3	150	105	45	100
Pima	Diabetes (si/no)	8	2	336	235	101	76
Desord. cardiac.	Corazón (si/no)	13	2	270	189	81	77.5 y 81
Peterson Barney	Voz	3	2	275	192	83	- -
Proteinas	Localiz. de prot.	7	8	336	235	101	81
Vino	Vino	13	3	178	125	53	96 al 100

Tabla 7.5: Características relevantes de las bases de datos.

El número de ejemplos de entrenamiento corresponde al 70% de los ejemplos totales y el 30% restante corresponde a los ejemplos de prueba.

La Tabla 7.6 muestra la distribución de los ejemplos en las clases.

En la tabla 7.7 se reportan los parámetros obtenidos por medio de experimentación para el programa evolutivo propuesto: el tamaño de la población para cada base de datos varió en 10, 15 y 20, y el número de generaciones para cada base de datos fueron de 100, 150, 200, 250, 500 y 1000, obteniéndose los mejores resultados en el tamaño de la población y número de generaciones

Base de datos	Distribución de los ejemplos en las clases
Iris	50 ejemplos para cada clase
Pima	111 clase diabetes negativa y225 clase diabetes positiva
Desord. card.	150 clase sanos y 120 clase enfermos
Peterson Barney	132 clase UH y 143 clase UW
Proteínas	Mínimo 2 ejemplos, máximo 143 son 8 clases
Vino	59 ejemp. Clase 1, 71 ejemp. Clase 2 y 48 ejemp. Clase 3

Tabla 7.6: Número de ejemplos de las bases de datos.

reportadas; cabe mencionar que a veces no era necesario hacer todas las combinaciones del número de generaciones y tamaño de la población ya que con la experimentación a veces era muy claro cuáles eran los parámetros idóneos.

El número máximo de tamaño de los cromosomas se obtuvieron también a base de experimentación, basándonos en la complejidad de clasificación de cada base de datos se hicieron pruebas con diferentes tamaños obteniéndose mejores resultados en los reportados.

La última columna reporta el tiempo que tardo el programa evolutivo en ejecutarse con los parámetros indicados; el programa se ejecutó en una PC pentium, con 32 megas de RAM bajo ambiente Windows.

BD	Población	Gener.	Max. tam. cromosom.	Tiempo ejecuc.
Iris	10	150	15	25 seg
Pima	15	200	15	60 seg
Desord. card.	10	150	50	10 seg
Peterson Barney	10	500	100	40 seg
Proteínas	10	500	50	190 seg
Vino	10	350	50	50 seg.

Tabla 7.7: Tiempos de ejecución del programa evolutivo.

En la tabla 7.8 se reportan los porcentajes de aciertos de los ejemplos de entrenamiento y de los de prueba, además de la aptitud final calculada y del número de nodos ocultos encontrado por el programa evolutivo en cuestión, incluyendo nuevamente los datos de los porcentajes de aciertos reportados en otros trabajos utilizando otros algoritmos para clasificación.

BD	Entran. aciertos	Prueba % aciertos	Aptitud	Nodos Ocult.	% aciert. reportados
Iris	90.4	**100**	0.0869	6	**100**
Pima	80.85	**74.25**	0.0315	5	**76**
Desord. card.	83	**77.7**	0.0493	9	**77.5 y 81**
Peterson Barney	89.07	**78.26**	0.0650	20	-
Proteínas	68.9	**63.6**	0.0125	31	**81**
Vino	92.8	**98.11**	0.1215	18	**96 al 100**

Tabla 7.8: Porcentajes de acierto del programa evolutivo.

Es importante notar que en los resultados reportados para la base de datos de los desordenes cardíacos, siempre se trabajo un algoritmo genético con longitud fija equivalente a 13 nodos ocultos, y también recordar que el programa evolutivo nos proporciona el número de nodos ocultos y los pesos de las conexiones del PMC. Vemos que, con la excepción de la base de datos Proteínas, los resultados son bastante competitivos.

7.6 Observaciones Finales

En el capítulo hemos visto diferentes maneras de incorporar ideas genéticas en algoritmos computacionales para encontrar soluciones a problemas difíciles de optimización. Los dos enfoques principales que revisamos fueron los algoritmos genéticos, de Holland, y las estrategias evolutivas, de Bienert, Rechenberg, y Schwefel. Ambas técnicas tienen puntos importantes en común:

1. Manejan poblaciones de individuos.

2. Cada individuo es la codificación de una solución al problema.

3. Hay una función de aptitud definida sobre los individuos, y que indica que tan bueno es cada uno de los individuos.

4. El problema de optimización a resolver está íntimamente relacionado con la función de aptitud.

5. La aplicación de operadores genéticos como selección, cruzamiento y mutación.

6. La iteración de una población a otra formada aplicando los operadores genéticos.

Sin embargo, hay diferencias entre las técnicas que se manifiestan con:

1. Los individuos en los algoritmos genéticos son cadenas binarias, mientras que en las estrategias evolutivas son vectores con valores reales, usualmente más naturales para el problema.

2. Mutación para los algoritmos genéticos es de inversión de bits, mientras para las estrategias evolutivas es perturbación gaussiana. Se considera más importante la mutación para las estrategias evolutivas que para los algoritmos genéticos.

3. El cruzamiento es usualmente más importante para los algoritmos genéticos que para las estrategias evolutivas.

4. La auto-adaptación en las estrategias evolutivas permite cambios de parámetros.

Estas observaciones no deben ser entendidas como inmutables, debido a que desde sus inicios, se han ido desarrollando representaciones y operadores distintos a los originales, como vimos en 7.3.6 y 7.5.

Al igual que en el caso de las redes neuronales, el esquema puede parecer excesivamente sencillo comparado con el punto de vista biológico, sin embargo ha permitido soluciones a problemas complejos que no son fáciles de resolver con otras técnicas, por ejemplo cuando la función a optimizar no es continua. Además, la utilización de estas técnicas permite una programación en paralelo lo cual agiliza la ejecución de las instrucciones y, por lo tanto, mejoran los tiempos de ejecución [23].

Finalmente, uno podría preguntar ¿porqué funcionan? Intuitivamente, los puntos en común mencionados anteriormente permiten una exploración del espacio de búsqueda, así como una explotación de la mejor solución encontrada. En el caso de los algoritmos genéticos, el teorema del esquema de

Holland [51] es frecuentemente citado como la base para explicar el poder de los algoritmos genéticos, aunque también ha sido sujeto a criticas, c.f. [8], y formas alternas de modelar y analizar algoritmos genéticos con cadenas de Markov han sido empleados [45]. Para estrategias evolutivas, ya hemos mencionado algunos resultados teóricos como en [52].

Vale la pena concluir con la observación de que ningún algoritmo es 'infalible' y puede resolver de manera mejor a cualquier problema, pero los algoritmos que incorporan ideas genéticas tienen un cierto encanto, y han sido bastantes éxitos para una variedad interesante de problemas.

7.7 Ejercicios

1. Se quiere minimizar la función con un algoritmo genético:

$$f(x) = x^2/4000 - cos(x) + 1 \quad \text{para} \quad x \in [-30, 30].$$

¿ Cuál sería una posible función de aptitud cuando se utiliza el pseudocódigo de 7.3.2 con la selección dada por el método de la ruleta?

Además, codificar cada cromosoma con una cadena binaria suficiente para obtener una precisión de 8 dígitos después del punto decimal.

También, minimizar la función con un algoritmo genético:

$$f(x, y, z, w) = (x + 10y)^2 + 5(z - w)^2 + (y - 2z)^4 + 10(x - w),$$

para $-2 < x, y, z, w < 2$.

2. Repetir 1. utilizando estrategias evolutivas.

3. Escoge uno de los problemas de la sección 3.1 y aplica un algoritmo genético para su resolución. En un reporte, explica tus razones para elegir la codificación del problema, el tamaño de la población, el criterio de terminación, los tipos de mutación y cruzamiento y sus probabilidades, así como el método de selección. Deberás reportar veinte repeticiones del experimento.

4. Escoge uno de los problemas de la sección 3.2 y aplica un algoritmo genético para su resolución. En un reporte, explica tus razones para elegir la codificación del problema, el tamaño de la población, el criterio de terminación, los tipos de mutación y cruzamiento y sus probabilidades, así como el método de selección. Deberás reportar veinte repeticiones del experimento.

5. Escoge uno de los problemas de la sección 3.2 y aplica una estrategia evolutiva para su resolución. En un reporte, explica tus razones para elegir el tipo de estrategia, el tamaño de la población, el criterio de terminación, el tipo de mutación y sus parámetros. Deberás reportar veinte repeticiones del experimento.

6. Un robot puede moverse al norte (N), sur (S), este (E) u oeste (O); su comportamiento está determinado por una lista de qué movimientos debe tomar dentro del laberinto. Si pega contra una pared, entonces este movimiento se pierde. El laberinto tiene una entrada, donde empieza el robot, y una salida, a la que trata de llegar. Mientras más cerca llegue el robot a la salida en el menor número de movimientos, mejor será la calificación que obtengan.

   Se representa un robot por una lista de movimientos:

   (N E E S S O)

   El laberinto puede representarse como sigue:

```
# # # # # # # #
#             s #
# # #     # # # #
# e             #
#     L         #
# # # # # # # #
```

   Donde s es la salida, e es la entrada, y L es donde el robot esta localizado. Si ejecutamos los movimientos señalados, empezando en e,

terminamos en L. Como el robot no puede ver, se mueve hasta que llega a la salida s, y después ignora los demás movimientos en la lista, o bien termina sus movimientos sin llegar a la salida. Su cercanía a la salida determina su aptitud. Si llega a la salida, tiene una aptitud alta, y además, si lo hace en pocos movimientos, tiene una aptitud mayor.

Modelar el problema con un algoritmo genético explicando, en particular, tus razones para tu elección de la codificación de los cromosomas y tu definición de la función de aptitud.

Bibliografía

[1] Aarts, E. and Korst, J. (1987), *Boltzmann Machines and their Applications*, Lecture Notes in Computer Science 258, Springer-Verlag, pp. 34-50.

[2] Aarts, E. and Korst, J.(1989), *Simulated Annealing and Boltzmann Machines*, Wiley.

[3] Aeberhard S., Coomans D. , Vel O., (1992a), *Comparison of Classifiers in High Dimensional Settings*, Tech. Rep. no. 92-02, Dept. of Computer Science and Dept. of Mathem.

[4] Aguilar, J., (2003), *A set of experiments for the Combinatorial Ant System*, Technical Report, CEMISID 12-2003, Universidad de los Andes, Venezuela.

[5] Aguilar, J., Velásquez, L. and Pool, M. (2004), *The Combinatorial Ant System*, Applied Artificial Intelligence Journal, 18:5, pp. 427-446.

[6] Aguilar, J. (2005), *The combinatorial ant system for dynamic combinatorial optimization problems*, Revista de Matemática: Teoría y Aplicaciones, 12:(1-2), pp. 5160.

[7] Aho, A.V., Hopcroft, J.E. and Ullman, J.D., (1974), *The Design and Analysis of Computer Algorithms*, Addison-Wesley.

[8] Altenberg L., (1995), *The Schema Theorem and Price's Theorem*, In *Foundations of Genetic Algorithms 3*, ed. Darrell Whitley and Michael Vose, Morgan Kaufmann, San Francisco. pp. 23-49.

[9] Angeline P., Saunders G., Pollack J., (1994), *An Evolutionary Algorithm that Constructs Recurrent Neural Networks*, IEEE Transactions on Neural Networks, 5, pp. 54-65

[10] Antoniadis, A.; Berruyer, J.; Carmona, R. (1992) *Régression Non Linéaire et Applications.* Economica, Paris.

[11] Applegate D.L., Bixby R.E., Chvátal V. y Cook W.J., (2006), *The Traveling Salesman Problem: A Computational Study*, Princeton University Press.

[12] Back, T., Hoffmeister, F., and Schwefel, H-P., (1991), *A survey of evolution strategies*, Proceedings of the Fourth International Conference on Genetic Algorithms, pp. 2-9.

[13] Barnes J. W. and Laguna M. (1993), *A Tabu Search Experience in Production Scheduling*, Annals of Ops. Res., 41, pp. 141-156.

[14] Bates, D.M. and Watts, D.G. (1988), *Nonlinear Regression Analysis and its Applications.* John Wiley and Sons, Stateplace, New York.

[15] Barnes, J.W. and Vanston L. K. (1981), *Scheduling Jobs with Linear Delay Penalties and Sequence Dependent Setup Costs.* ORSA 29:1, pp.146-159.

[16] Binder, K. (1978), *Monte Carlo Methods in Statistical Physics*, Springer-Verlag.

[17] Bonabeau, E., Dorigo, M. and G. Theraulaz (1999), *Swarm Intelligence: from Natural to Artificial Swarm Systems.* Oxford University Press: USA.

[18] Bonomi, E. and Lutton, J.L. (986), *The Asymptotic Behavior of Quadratic Sum Assignment Problems: A Statistical Mechanics Approach*, European Journal

[19] Burkard, R.E. and Bnniger, T. (1983), *A Heuristic for Quadratic Boolean Programs with Applications to Quadratic Assignment Problem*, European Journal of Operational Research 13, pp. 374-386.

[20] Burkard, R.E. and Derigs, U. (1980), *Assignment and Matching Problems: Solution Methods with FORTRAN-Programs*, Springer-Verlag.

[21] Burkard, R.E. and Stratmann, K. (1978), *Numerical Investigations on Quadratic Assignment Problems*, Naval Research Logistics Quarterly 25, pp. 129-148.

[22] Burkard,R.E. (1990), *Locations with Spatial Interactions: The Quadratic Assignment Problem*, Discrete Location Theory, eds. Mirchandani y Francis, Wiley.

[23] Castro M., Roman G., Buenabad J., Martinez A., Goddard J., (2004), *Integration of load balancing into a parallel evolutionary algorithm*, Lecture Notes in Computer Science Volume 3061, Advanced Distributed Systems, Springer, pp.219-230.

[24] Chakrapani, J. and Skorin-Kapov, J. (1993), *Massively Parallel Tabu Search for the Quadratic Assigment Problem*, Annals of Ops. Res., 41, pp. 327-341.

[25] Christofides, N. (1975), *Graph Theory: An Algorithmic Approach*, Academic Press.

[26] Coello Coello, Carlos A. (1995), *Introducción a los Algoritmos Genéticos. Soluciones Avanzadas*, Tecnologías de Información y Estrategias de Negocios, 3: 17, pp. 5-11.

[27] Colbourn, C. (1984), *The complexity of completing partial latin squares*, Discrete Applied Mathematics, 8, pp. 25-30.

[28] Committee on the Next Decade of Operations Research (Condor 1988), *Operations Reseach: The Next Decade*, Ops. Res., 36, pp. 1-15.

[29] Corne, D., Dorigo, M. and Glover, F. (1999), *New ideas in Optimization.* McGraw Hill, Holland.

[30] Crescenzi P. y Kann V. (Eds), *A compendium of NP optimization problems*, http://www.nada.kth.se/~viggo/problemlist/compendium.html.

[31] Dasarathy B.V., (1980), *Nosing Around the Neighborhood: A New System Structure and Classification Rule for Recognition in Partially Exposed Environments*, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-2, No.1, pp. 67-71.

[32] de-los-Cobos-Silva.S., Gutiérrez Andrade M. y Pérez Salvador B., (1995) *Lineamientos de Implantación de la Búsqueda Tabú para los Problemas de Calendarización*, Memorias del IX Simposio Métodos Matemáticos Aplicados a las Ciencias, Turrialba, Costa Rica. pp. 23-32.

[33] de-los-Cobos-Silva S. G. (1996), *Optimización combinatoria en la industria del petróleo*, VI Escuela nacional de optimización y análisis numérico, Puebla, México.

[34] de-los-Cobos-Silva S., González-Santoyo F., Goddard Close J. y Pérez-Salvador B.(2002), *Optimización por Enjambre de Partículas Aplicada a Inventarios Multiproducto*, IX SIGEF CONGRESS, nov. 25-27, Mérida, Venezuela.

[35] de-los-Cobos-Silva S., Pérez Salvador B.R., Gutiérrez Andrade M. A. y Ordorica Mellado M. (1997), *Búsqueda Tabú: Metaprocedimiento de Mejora para el Problema de Inventarios Multiproducto*, Memorias del X Simposio Internacional de Métodos Matemáticos Aplicados a Las Ciencias Liberia, Costa Rica. W. Castillo y J. Trejos (Eds.), pp: 97-106.

[36] de-los-Cobos-Silva S., Pérez-Salvador B. y Gutiérrez Andrade M. (1996), *Programación Estocástica en Optimización*. IMSIO-DECFI, Universidad Nacional Autónoma de México, México.

[37] de-los-Cobos-Silva, S. (1994), *La Técnica de la Búsqueda Tabú y sus Aplicaciones*, Tesis de Doctorado, Posgrado de Ingeniería, UNAM, México.

[38] de-los-Cobos-Silva S.G., Goddard Close J.,Pérez-Salvador Blanca Rosa y Trejos Zelaya J. (2003), *Regresión No Lineal Mediante Optimización por Enjambre de Partículas*. Memorias XVII Foro Nacional de Estadística, pp. 39-45.

[39] Draper, N.R. and Smith, H. (1968), *Applied Regression Analysis.* John Wiley and Sons, Stateplace, New York.

[40] Dorigo, M. (1992), *Optimization, Learning and Natural Algorithms.* Ph.D Thesis, Politecnico de Milano, Italy.

[41] Dorigo, M. and Gambardella, L. (1997), *Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem.* IEEE Trans. on Evolutionary Computation, Vol. 1, pp. 53-66.

[42] Dorigo, M., Maniezzo V. and Coloni, A. (1996), *The ant system: Optimization by a colony of cooperating agents.* IEEE Trans. Syst. Man, Cybern., Vol. 26, pp. 29-41.

[43] Droste, S., Jansen, T., and Wegener, I., (2002), *On the Analysis of the (1+1) Evolutionary Algorithm*, Theoretical Computer Science 276, pp. 51-81.

[44] Eiben, A.E., Smith, J.E., (2007), *Introduction to Evolutionary Computing*, Springer, Natural Computing Series, Corr. 2nd printing.

[45] Fogel D.B., (1994), *Asymptotic Convergence Properties of Genetic Algorithms and Evolutionary Programming: Analysis and Experiments*, Cybernetics and Systems, Vol. 25:3, pp. 389-407.

[46] ftp://svr-ftp.eng.cam.ac.uk/pub/comp.speech/data/

[47] Glover F. (1989), *Tabu Search, Part I*, ORSA Journal on Computing, 1:3, pp. 190-206.

[48] Glover F. (1990a), *Tabu Search, Part II*, ORSA Journal on Computing, 2:1, pp. 4-31.

[49] Glover F. and Laguna M. (1993), *Tabu Search, Modern Heuristic Techniques for Combinatorial Problems*, Colin R. Reeves(Ed.), Blackwell Scientific Publications, Oxford, pp. 70-150.

[50] Goddard J., López I., Rufiner L., (1996), *Diagnóstico de cardiopatías mediante redes neuronales y algoritmos genéticos*, Revista Argentina de Bioingeniería, Vol.2, No.2, pp. 11-19.

[51] Goldberg D.E., (1989), *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley Professional.

[52] Greenwood G.W. and Zhu Q., (2001), *Convergence in Evolutionary Programs with Self-Adaptation in Evolution Strategies*, Evolutionary Computation 9, pp. 147-157.

[53] Gutiérrez Andrade, M.A. (1991), *La Técnica de Recocido Simulado y sus Aplicaciones*, Tesis Doctoral, División de Estudios de Posgrado de la Facultad de Ingeniería, UNAM, México.

[54] Hancock, P.J.B., (1994), *An empirical comparison of selection methods in evolutionary algorithms*, Lecture Notes in Computer Science 865, 80-94, Fogarty, T.C. (Ed).

[55] Hassoun M. H., (1995), *Fundamentals of artificial Neural Networks*, The MIT Press.

[56] Hertz, A., Kobler, D. (2000), *A Framework for the Description of Evolutionary Algorithms*, European Journal of operational Research, **126**, pp. 1-12.

[57] Hidrobo, F. and Aguilar, J. (1998), *Toward a Parallel Genetic Algorithm Approach Based on Collective Intelligence for Combinatorial Optimization Problems*, In Proc. IEEE International Conference on Evolutionary Computation, pp. 715-720.

[58] Hinterding R., Gielewski H., Peachey T.C., (1995), *The nature of mutation in genetic algorithms*, Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA), pp. 65-72.

[59] Horton P. & Nakai K., (1996), *A Probablistic Classification System for Predicting the Cellular Localization Sites of Proteins*, Intelligent Systems in Molecular Biology, pp. 109-115.

[60] Hu, T.C. (1982), *Combinatorial Algorithms*, Addison-Wesley.

[61] Johnson, D.S., placecountry-regionAragon, C.R., McGeoch, L.A. and Schevon, C. (1989), *Optimization by Simulated Annealing: an Experimental Evaluation; Part I, Graph Partitioning*, Operations Research 37, pp. 865-892.

[62] Judd J. S., (1990), *Neural Network Design and the Complexity of Learning*, Cambridge, MA, MIT Press.

[63] Karr C. & Freeman L.M., (Eds.) (1998), *Industrial Applications of Genetic Algorithms*, CRC.

[64] Kaspi M. and Rodenblatt M.J. (1985), *The Effectiveness of Heuristic Algorithm for Multi-Item Inventory Systems with Joint Replenishment Costs.* International Journal of Production Research 23, pp. 109-116.

[65] Kennedy, J. and Eberhart, R.C. (2000), *Intelligent Swarm Systems.* Academic Press, New York.

[66] Kennedy, J., Eberhart, R.C. and Shi Y. (2001), *Swarm Intelligence.* Morgan Kaufmann.

[67] Kirkpatrick, S., Gelatt C.D. and Vecchi, M.P. (1983), *Optimization by Simulated Annealing*, Science 220, pp. 671-680.

[68] Koopmans.T.C. and Beckmann, M.J., (1957), *Assignment Problems and the Location of Economic Activities*, Econometrica 25, pp. 53-76.

[69] Kuijpers C. M. H., Murga R. H., Inza I. and Dizdarevic S., (1999), *Genetic algorithms for the travelling salesman problem: A review of representations and operators*, Artificial Intelligence Review, Volume 13, Issue 2, pp. 129-170.

[70] Laarhoven P.J.M. van and Aarts E.H.L. (1988), *Simulated Annealing: Theory and Applications*, Reidel.

[71] Laguna M. (1993), *A Guide to Implementing Tabu Search*, Technical report (October), Graduate School of Business and Administration, University of Colorado, Boulder.

[72] Laguna M., Barnes J. W. and Glover F. (1990), *Tabu Search for a Single Machine Scheduling Problem*, Technical report (july), Advanced Knowledge Systems Group of U S, West Advanced Technologies.

[73] Laguna, M. and Kelly, J. (1993), *Master Production Scheduling in a Single Facility with Sequence-Dependent Changover Times.*, Technical report (september), Graduate School of Business and Administration, University of Colorado at Bould.

[74] Love, R.F., Morris, J.G. and Wesolowsky, G.O. (1988), *Facilities Location Models and Methods*, North-Holland.

[75] Lynce, I., Ouaknine, J. (2006), *Sudoku as a SAT problem*, Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics, Springer.

[76] Marquardt, D.W. (1963), *An algorithm for least squares estimation of nonlinear parameters*, Journal of the Society for Industrial and Applied Mathematics 2: 431-441.

[77] Martínez Licona, A. (2002), *Definición de una red neuronal perceptron por medio de un programa evolutivo con cromosomas de longitud variable para la solución de problemas de clasificación*, Maestría en Ingeniería Biomédica, U.A.M.-I.

[78] Martínez Licona A.E. y Goddard J., (2001), *Definición de una red neuronal para clasificación por medio de un programa evolutivo*, Revista Mexicana de Ingeniería Biomédica, Vol XXII Núm. 1, pp. 4-11.

[79] Metropolis, N., Rosenbluth, M., Rosenbluth, A.,Teller, A. and Teller, E.,(1953) *Equation of state calculations by fast computing machines*, Journal of Chemical Physics 21, pp. 1087-1092.

[80] Michalewicz Z., (1998), *Genetic Algorithms + Data Structures = Evolution Programs*, Springer, 3ra ed.

[81] Mirchandani, P.B. and Francis, R.L. editores (1990), *Discrete Location Theory*, Wiley.

[82] Mirchandani, P.B. and Reilly, J.M. (1986), *Spatial Nodes in Discrete Location Problems*, Annals of Operations Research 6, pp. 203-222.

[83] Molga and Smutnicki, M.C. (2005), *Test functions for optimization*, disponible en www.zsd.ict.pwr.wroc.pl/files/docs/functions.pdf

[84] Montana D., (1995), *Neural Network Weight Selection Using Genetic Algorithms*, capitulo 5 de Intelligent Hybrid Systems, Goonatilake and Khebbal (Eds.), John Wiley & Sons.

[85] Moraglio, A., Togelius, J., (2007), *Geometric particle swarm optimization for the Sudoku puzzle*, Proceedings of the 9th annual conference on Genetic and evolutionary computation, pp. 118-125.

[86] Muñoz M. A., López J. A., y Caicedo E. F., (2008), *Inteligencia de enjambres: sociedades para la solución de problemas(una revisión)*, Revista Ingenieriía e Investigación, 28, pp. 119-130.

[87] Narro Ramírez A.E. and de-los-Cobos-Silva, (1994), *Algorithms for Multi-item Inventory Systems*, 15 th. International Symposium on Mathematical.

[88] Nemhauser, G.L., and Wolsey, (1989), *Integer and Combinatorial Optimization*, Wiley.

[89] Nugent, C.E., Vollmann, T.E. and Ruml, J., (1968), *An Experimental Comparison of Techniques for Assignment of Facilities to Locations*, Operations Research 16, pp. 150-173.

[90] Osman, placeI. and Potts, CN. (1989), *Simulated Annealing for Permutation Flow-Shop Scheduling*, Omega 17, pp. 551-557.

[91] Papadimitriou, C.H. and Steiglitz, K., (1982), *Combinatorial Optimization: Algorithms and Complexity* , Prentice-Hall.

[92] Raidl G. R., (2005), *Evolutionary computation: An overview and recent trends*, OGAI Journal, 24, pp. 2-7.

[93] Ratkowsky, D.A. (1983), *Nonlinear Regression Modeling: A Unified Practical Approach*, Marcel Dekker, Inc., Stateplace, New York.

[94] Reeves, C. (Ed.), (1995), *Modern Heuristic Techniques for Combinatorial Problems.* McGraw-Hill, London.

[95] Rudolph G., (1994), *Convergence analysis of canonical genetic algorithms, IEEE Transactions on Neural Networks*, 5 pp. 96-101.

[96] Rumelhart, D.E., Hinton, G.E., and Williams, R.J. (1986), *Learning representations by back-proating errors.* Nature, 323, 533-536

[97] Sarker R. Mohammadian M. & Yao X., (Eds.), (2002), *Evolutionary Optimization*, Springer.

[98] Sima J.,(2002), *Training a single sigmoidal neuron is hard*, Neural Computation, 14(11), pp. 2709-2728.

[99] Smith J.W., Everhart J.E., Dickson W.C., Knowler W.C. and Johannes R.S., (1988), *algoritmo de aprendizaje ADAP para diagnósticar la diabtes mellitus*, proceedings of the Symposium of computer applications and medical care, pp. 261-265.

[100] Syslo, M., Deo, N. and Kowalik, J. (1983), *Discrete Optimization Algorithms with PASCAL Programs*, Prentice-Hall.

[101] Toda, M., Kubo, R. and Sait, N. (1983), *Statistical Physics*, Springer-Verlag.

[102] Tomassone, R., Audrain, S., and Lesquoy, E.; Millier, C. (1992), *La Régression. Des Nouveaux Regards sur une Ancienne Méthode Statistique.* Masson, Paris.

[103] Tovey C., (2002), *Tutorial on Computational Complexity*, Interfaces 32:3, 30-61.

[104] Trejos, J. and Villalobos, M. (1999), *Optimización mediante recocido simulado en regresión no lineal*, Memorias del XII Foro Nacional de Estadística, Monterrey, pp. 183–190.

[105] Trejos, J., Goddard, J., Piza, E. y de-los-Cobos, S. S.(2002), *Clasificación de Datos Numéricos Mediante Optimización por Enjambre de Partículas.* 5th. International Conference of Operations Research, March 4-8, Havana, Cuba.

[106] Trejos, J., Murillo, A. and Piza, E. (1998), *Global stochastic optimization for partitioning*, In: A. Rizzi et al. (Eds.), Advances un Data Science and Classification, Springer-Verlag, StateBerlin.

[107] Trejos-Zelaya, J., de-los-Cobos, S. S.and Villalobos, M. (2001), *Aplicación de la Búsqueda Tabú en Regresión No Lineal.* VIII Congreso Latinoamericano de Probabilidades y Estadística Matemática, 12-16 de noviembre, La Habana, Cuba.

[108] Trejos-Zelaya, J., *Regresión*, (por publicarse), Universidad de Costa Rica.

[109] UCI Repository of Machine Learning Databases and Domian Theories, ftp://ics.uci.edu.pub/machine-learning-databases

[110] Villalobos, M., Trejos J. y de-los-Cobos-Silva S. (2006), *Aplicación de la Búsqueda Tabú en Regresión no lineal*, Revista de Matemática: Teoría y Aplicaciones, 13(1), pp. 81-94.

[111] Xin Yao, (1996a), *An Overview of Evolutionary Computacion*, Chinese Journal of Advanced Software Research, Vol 3, No. 1

[112] Xin Yao, Young Liu., (1999a), *Towards Designing Artificial Neural Networks by Evolution*

[113] Xin Yao, (1999a), *Evolving artificial Neural Networks*, Proc. of the IEEE, 87; pp 1423-1447

[114] Yao X. and Xu Y., (2006), *Recent Advances in Evolutionary Computation*, Journal of Computer Science and Technology, **21**, pp.1-18.

[115] Yato T. y Seta T., (2002), *Complexity and completeness of finding another solution and its application to puzzles*, In Proceedings of the National Meeting of the Information Processing Society of Japan (IPSJ).

Índice de Materias

2-cambio, 27

adaptación determinísta, 218
agujero negro, 106, 133
algoritmo (μ + λ)-ES, 214
algoritmo (μ,λ)-ES, 214
algoritmo (1+1)-ES auto-adaptativo, 219
algoritmo de Metropolis, 66
algoritmo de retropropagación, 48
algoritmo de solución, 27
algoritmo eficiente, 33
algoritmo exacto, 27
algoritmo genético, 198
algoritmo polinomial, 33
algoritmos (1 + 1)-ES, 215
algoritmos de orden O($n\hat{2}$), 21
algoritmos estocásticos, 21
algoritmos evolutivos, 189
algoritmos genéticos, 189, 191
algoritmos heurísticos, 21, 30
algoritmos metaheurísticos, 30
Ant System, 176
Ant-cycle, 177
Ant-density, 177
Ant-quantity, 177
Aprendizaje en Perceptrones Multicapa, 46
aptitud acumulada, 227
aptitud relativa, 227
atributos por movimiento de intercambio, 122
auto-adaptación, 218

búsqueda local, 23, 28, 69
búsqueda tabú, 105
baño térmico, 65
back-propagation, 48
Bin Packing Problem, 41
buenas soluciones, 31

calendario, 120
calidad de solución, 75
capa de entrada, 47
capa de salida, 47
capa intermedia, 47
capa oculta, 47
Chakrapani, 148
cláusulas, 43
clan, 42
clase cromática, 40
clase de color, 40
clase P, 33
coloración de gráfica, 40
coloración de una gráfica, 40
conexiones entre nodos, 47
conjunto de entrenamiento, 48
conjunto de soluciones posibles, 23
conjunto estable, 39
conjunto factible, 23
conjunto independiente, 39

conjunto independiente máximo, 39
conjunto independiente maximal, 39
constante de Boltzmann, 65
convergencia de recocido simulado, 69
criterio de aceptación, 67
criterio de aspiración, 110, 115, 119, 123, 125, 137, 146
criterio de Metropolis, 66
criterio de paro de BT, 123
criterio de terminación, 199
criterios de aspiración, 137
cromosoma de longitud variable, 222
cruzamiento, 189, 195
Cuadrados Mágicos, 44
cubrimiento por vértices, 41

distribución de Boltzmann, 64
diversificación regional, 116

eficiencia del algoritmo, 75
elitismo, 199
EMC, 227
entrampamientos suboptimales, 133
equilibrio térmico, 64, 66
error cuadrático medio, 227
espacio de soluciones, 26
estado amorfo, 65
estado fundamental, 65
estrategia de oscilación, 130
estrategias evolutivas, 189, 190, 213
estructura de vecindades, 27
explosión combinatoria, 19

fórmula booleana, 43
fórmula satisfacible, 43
factor de Boltzmann, 65
feromona, 177
función costo, 26
función costo/beneficio, 23
función de activación, 47
función de partición, 64
función de Shubert, 24
función de tranferencia, 223
función objetivo, 23, 26
función sigmoide, 47

gráfica coloreable, 40
gradiente descendiente, 223

heurísticas, 30

instancia, 25
instancia de un problema , 29
inteligencia de enjambre, 160, 163
intensificación regional, 116
Intensificación y diversificación regional, 132
inventarios multiproducto, 167

k-cambio, 27
k-coloración, 40
Knapsack Problem, 44

la memoria de término corto, 113
la regla de éxito 1/5, 218
lista tabú, 109, 119
loading problem, 48

máximo local, 25, 29
Método Hormiga, 176
método de búsqueda tabú, 118
método de la ruleta, 193
método de reproducción, 193
métodos de selección, 193
mínimo local, 25, 29
mínimos cuadrados, 172
múltiples padres, 221
Magic Squares, 44
manejo dinámico de la lista tabú, 147

matriz de frecuencias, 125
matriz tabú, 125
Maximum Clique, 42
mecanismo de generación, 28, 67
mecanismo de transición, 69
memoria de plazo corto, 120, 122, 123
memoria de término corto, 110
memoria de término intermedio y largo, 131
memoria flexible, 107, 108
meta de las restricciones, 122
metaheurísticas, 30
movimiento admisible, 113, 119, 122
movimiento candidato, 113, 122
movimiento de intercambio, 121
movimiento de partícula, 164
movimiento tabú, 109
movimientos tabú, 119
mutación, 189, 197
mutación estructural, 228
mutación paramétrica, 228

número cromático, 41
número de estabilidad, 39
número independiente, 39
neurona, 47
nivel de aspiración, 115
notación O-grande, 34

operador cruzamiento, 192
operador mutación, 192
operador selección, 192
operadores de búsqueda, 189
operadores genéticos, 190, 192
optimalidad local, 28
optimización combinatoria, 19
optimización continua, 19, 23
optimización discreta, 19, 23
optimización por enjambre de partículas, 160
óptimo local, 25, 29
ordenamiento por selección, 34

parámetro de control, 66
partículas, 162
penalización, 136
Perceptrones Multicapa, 46
permutación cíclica, 26, 42
peso, 47
problema de aprendizaje, 48
Problema de asignación cuadrática, 143
problema de asignación cuadrático, 36, 38, 77
Problema de Asignación Tridimensional Axial, 46
problema de calendarización, 120, 124
Problema de Clan Máximo., 42
Problema de Coloración Mínima, 40
Problema de Conjunto Independiente, 39, 71
Problema de Corte Máximo., 42
Problema de Cubrimiento de Vértices, 41
problema de empaquetamiento, 41
problema de entrenamiento, 48
Problema de la Mochila, 44
Problema de Satisfacibilidad, 43
problema de Sudoku, 45
problema del agente viajero, 35, 42
problema polinomial, 33
problema polinomial no-determinista, 35
problema QAP, 38
problemas de optimización, 19
problemas intratables, 21
problemas NP, 33, 35

problemas NP-Completos, 35
problemas NP-Duros, 21
problemas P, 33
problemas P y NP, 33
programa de enfriamiento, 69, 70
programación dinámica, 64
programación genética, 189
programación lineal, 63
pseudo-código de Búsqueda Local., 29
pseudocodigo de recocido simulado., 68
PSO, 160

recocido de un sólido, 64
recocido simulado, 63
recorrido, 26
regresión no lineal, 171
reproducción, 193
restricciones tabú, 114
retropropagación, 223
ruleta, 193

satisfacible, 43
sigmoide, 47
simulated annealing, 63
sistema hormiga, 176
solución globalmente óptima, 24, 26
solución en tiempo polinomial, 20
soluciones élites, 117

The Vertex Cover Problem, 41
Three Dimensional Axial Assigment Problem, 46
transición, 67

umbral, 47

valle profundo, 106, 132
valor de movimiento, 112
valor de salida, 47
valor del movimiento, 121
variable booleana, 43
vecindad, 27
vecindad exacta, 29
vecindario, 118
vecino, 27
ventajas de la búsqueda local, 30

Búsqueda y Exploración Estocástica

Universidad Autónoma Metropolitana
Unidad Iztapalapa

Departamento de Ingeniería Eléctrica
División de Ciencias Básicas e Ingeniería

Impresión offset, en papel bond blanco de 90 grs., 268 páginas.
Portada en cartulina sulfatada 1 cara de 12 pts.
Medida final 17 x 23 cms., Encuadernación rústica cosida.

Concepto impreso
Fray Bernardino de Sahagún núm 99
Col. Vasco de Quiroga, Deleg. Gustavo A. Madero
CP 07440, México DF

La edición consta de 1 000 ejemplares

marzo 2010

Miguel Ángel Gutiérrez Andrade es Matemático de la UNAM, y obtuvo su maestría y doctorado en investigación de operaciones en la UNAM. Es profesor investigador la Universidad Autónoma Metropolitana desde 1986. Sus campos de interés incluyen optimización, métodos computacionales de gran escala, métodos heurísticos y modelos de equilibrio general computable.

Alma Edith Martínez Licona
Profesora Investigadora de la UAM Iztapalapa desde 1993 en el Departamento de Ing. Eléctrica. Licenciada en Computación con Maestría en Ciencias en Ingeniería Biomédica, ambos estudios realizados en la UAM I. Miembro del Cuerpo Académico Consolidado - Inteligencia Artificial Aplicada desde el 2006 (Institución: Secretaría de Educación Pública). Áreas de interés: Inteligencia Artificial (Redes Neuronales y Algoritmos Genéticos).

En las últimas décadas se han desarrollado varios métodos heurísticos para resolver problemas de optimización que anteriormente eran muy difíciles de resolver, si no es que desde el punto de vista práctico, imposibles de resolver. Este libro contiene información para resolver problemas usando algunas técnicas heurísticas tales como: recocido simulado, búsqueda tabú, inteligencia de enjambre, algoritmos genéticos y estrategias evolutivas. Cada uno de los capítulos termina con una sección de ejercicios que refuerzan el aprendizaje del capítulo y extienden la información tratada.

El libro está escrito en un estilo ágil y atractivo y está dirigido a estudiantes que se encuentran en los últimos trimestres de licenciatura o en posgrado, así como a profesionales que requieren saber técnicas de optimización para resolver problemas difíciles, en donde las técnicas clásicas fallan.

Cualquiera que lea y entienda el material de este libro, adquirirá una poderosa herramienta para resolver problemas de optimización.

ISBN 978-607-477-239-5

9 786074 772395