

Metaheurísticas para problemas de optimización restringidos

Guillermo Leguizamón



Departamento de Informática
Universidad Nacional de San Luis
Ejército de los Andes 950
(5700) San Luis
Argentina

e-mail: legui@unsl.edu.ar

Tesis para optar por el grado de
Doctor en Ciencias de la Computación

Asesores:

Dr. Zbigniew Michalewicz
University of North Carolina at Charlotte

Dr. Carlos Coello Coello
Departamento de Ingeniería Eléctrica
CINVESTAV-IPN, Mexico, DF.

*A mis padres y hermanos,
quienes de una u otra manera han sabido brindarme su apoyo.*

*A Raúl y Susana,
quienes a través de su ejemplo,
me enseñaron a transitar los innumerables caminos de la vida.*

Agradecimientos

La consecución de un trabajo de tesis doctoral no es el final de un camino, sino el comienzo de una nueva etapa con infinitas posibilidades en cuanto a la generación y difusión del conocimiento científico. Esta no es una tarea fácil, la cual requiere de mucha dedicación y compromiso. Sin embargo, poder llegar hasta este punto de la vida académica, es posible decir que con esfuerzo y empeño, se pueden alcanzar todas las metas propuestas.

En primer lugar, quiero expresar mi agradecimiento a la Universidad de San Luis y a todas las personas que son parte de ella, quienes a través de su trabajo diario hacen posible la formación académica en los distintos niveles. En mi caso particular, por brindarme los medios necesarios para llevar a cabo mis estudios de posgrado.

A mis asesores, el Prof. Zbigniew Michalewicz y el Prof. Carlos Coello Coello, quienes hicieron posible el desarrollo de esta tesis. En especial, mi agradecimiento al Prof. Coello Coello, quien a través de sus invalorable consejos y observaciones, me permitió recuperar la confianza para continuar y lograr concluir mis estudios de posgrado.

A Susana Esquivel, por ser mi guía indiscutible de mis primeros pasos en la docencia y por ser un modelo de superación, lucha y fortaleza ante los ineludibles obstáculos de la vida.

Mi más profundo agradecimiento al Prof. Raúl Gallard, de quien atesoro una infinidad de enseñanzas desde el punto de vista académico y humano. Mis deseos, al escribir este párrafo, son de mantener viva la memoria de una persona que dedicó gran parte de su vida a la superación académica en beneficio de nuestra Institución y de muchos colegas y estudiantes del Departamento de Informática.

Por último, pero no menos importante, mis más afectuoso agradecimiento a:

Marcelo Errecalde, con quien hemos compartido y espero, sigamos compartiendo, discusiones constructivas en relación a nuestros respectivos temas de investigación y de la vida en general.

A Oli, Norma y Tete, colegas y amigas, quienes me han brindado su afecto y apoyo en esta etapa de mi vida. En especial a Oli, mi compañera de “viaje” y de charlas inagotables,

A Beto, Jacquie y sus hijos, Mario y Manuel, por quienes tengo un gran afecto.

A todos los miembros del LIDIC de la Universidad Nacional de San Luis.

A mis colegas del Area de Autómatas y Lenguajes, especialmente a Patricia Roggero, con quien hemos recorrido juntos los intrincados y motivantes caminos de la Teoría de Computación y a Javier Apolloni, por su inmensa voluntad y espíritu de colaboración en el dictado de las materias a mi cargo.

Prólogo

La gran mayoría de los problemas del mundo real incluye algún tipo de restricciones, lo que implica que sólo es válido, un subconjunto del espacio de todas las posibles soluciones para dichos problemas.

La clase de problemas con restricciones varían según su dominio de aplicación (esto es, continuo o discreto); función objetivo; número y características de las restricciones a las que está sujeto. Sea continuo o discreto, el espacio de soluciones de un problema restringido se divide en regiones factibles y no factibles. Estas regiones están determinadas por el conjunto de restricciones respecto de su tamaño, forma y distribución en el espacio de soluciones.

Diseñar un algoritmo efectivo para realizar una búsqueda exitosa en un espacio altamente restringido, puede ser una tarea ardua y muchas veces imposible. La naturaleza de las restricciones puede hacer que las regiones factibles que contienen una solución óptima para el problema, sean muy difíciles de alcanzar.

Los problemas restringidos han sido ampliamente estudiados en el campo de las Matemáticas e Investigación Operativa. De hecho, existen numerosos algoritmos exactos para resolver muchos problemas restringidos. Sin embargo, la existencia de tales algoritmos está supeditada a que la función objetivo y las ecuaciones/inecuaciones que definen las restricciones, satisfagan ciertas propiedades; en la mayoría de los casos, muy estrictas.

Otra característica relevante de los problemas restringidos, independientemente de las restricciones a las que está sujeto, es que es altamente probable que una solución óptima se encuentre sobre la frontera entre la región factible y no factible del espacio de búsqueda. Esta situación puede implicar un mayor grado de dificultad en el proceso de búsqueda haciendo a un método específico mucho más costoso o bien, inviable para poder alcanzar una solución sobre la frontera entre la región factible y

no factible.

En los últimos años, han surgido nuevos enfoques para resolver problemas de búsqueda y optimización. Entre estos nuevos enfoques, están las metaheurísticas, a las que podemos definir en términos generales como técnicas de búsqueda que integran, de diferentes maneras, procedimientos de mejoras locales y estrategias de alto nivel para crear un proceso capaz de escapar de óptimos locales y realizar una búsqueda robusta en el espacio de soluciones.

Estos métodos no son exactos, aunque han mostrado a través de una gran cantidad de desarrollos y aplicaciones, ser efectivos para resolver una gran variedad de problemas del mundo real, incluyendo problemas altamente restringidos. En muchos casos, las metaheurísticas definen algoritmos que son representativos del estado del arte para importantes problemas académicos y del mundo real. Dentro de las metaheurísticas, existen aquellas que están basadas en una población, es decir, procesan una población de soluciones (o una población de agentes que generan soluciones) a fin de explorar el espacio de búsqueda. Otras, sólo utilizan una solución como punto de partida y exploran el espacio a través de la aplicación de operadores *ad hoc*.

En esta tesis se propone una técnica alternativa para el manejo de problemas con restricciones basado en lo que denominaremos a lo largo de este documento, como el “*Enfoque de frontera*”. Dicho enfoque será incorporado y probado en dos de las más difundidas metaheurísticas bio-inspiradas basadas en población: Algoritmos Evolutivos (AEs) y Algoritmos Basados en el Comportamiento de Colonias de Hormigas (ACO, sus siglas en Inglés para *Ant Colony Optimization* que será usada de aquí en adelante.).

Algunas características de los problemas con restricciones y experiencias previas justifican el desarrollo de métodos capaces de restringir la búsqueda a la frontera entre el espacio de búsqueda factible y no factible. Por ende, la propuesta incluye la incorporación del enfoque de frontera

en AEs y algoritmos ACO para su aplicación a problemas restringidos con dominio discreto y continuo. A los efectos de determinar el desempeño de los algoritmos implementados siguiendo el enfoque de frontera propuesto, se ha realizado un estudio experimental sobre un conjunto de problemas bien conocidos a fin de verificar su aplicabilidad. Dicho conjunto incluye una serie de problemas de programación no lineal (NLP: Non Linear Programming¹) con diferentes tipos de restricciones y dificultades y los siguientes problemas de optimización combinatoria: el problema de múltiples mochilas (MKP: Multiple Knapsack Problem), el problema de cobertura de conjuntos (Set Covering Problem) y el problema del conjunto independiente máximo (Maximum Independent Set Problem).

Como aporte adicional al enfoque de frontera propuesto, se plantea una adaptación a la meheurística basada en el comportamiento de hormigas a fin extender su aplicabilidad a otros tipos de problemas para las cuales fue definida originalmente.

¹Se usarán las respectivas siglas en Inglés de aquí en más.

Organización de la tesis:

- Capítulo 1:

Introduce algunas definiciones básicas de problemas de optimización, conceptos de algoritmos eficientes y una breve discusión sobre complejidad computacional. Este capítulo termina con una síntesis de los métodos más usados para resolver problemas de optimización.

- Capítulos 2 y 3:

Presentan una descripción de algunas de las metaheurísticas actuales más importantes: simulated annealing, tabu search, algoritmos evolutivos y basados en colonias de hormigas. Las dos primeras representativas de métodos basados en trayectoria usando un único punto de referencia. Las dos restantes, objeto de estudio en la presente tesis, son unas de las más representativas de las metaheurísticas basadas en población.

- Capítulo 4:

Desarrolla una discusión general sobre algunas de las más importantes técnicas representativas del estado del arte para manejo de restricciones usadas en el campo de la computación evolutiva y los algoritmos ACO.

- Capítulo 5:

Propone un enfoque de búsqueda en la frontera entre la zona factible y no factible para espacios continuos y discretos. Se presentan los detalles para cada una de las metaheurísticas consideradas, junto con la formulación de los problemas incluidos en el estudio experimental.

- Capítulos 6 y 7:

Presentan el estudio experimental detallado con los resultados obtenidos de la aplicación de algoritmos evolutivos y algoritmos ACO

respectivamente. Para ellos se consideró un conjunto importante de problemas duros de NLP y optimización combinatoria.

- Capítulo 8:

Realiza un análisis de algunos aspectos del comportamiento de los algoritmos evolutivos y algoritmos ACO, desde la perspectiva del enfoque de frontera y de las características propias de cada uno de ellos.

- Capítulo 9:

Finalmente, las conclusiones y futuras extensiones son consideradas en función de un análisis global de los resultados aportados en este trabajo.

Indice General

1. Introducción: Conceptos y Definiciones	1
1.1. Problemas de optimización	1
1.1.1. Optimización Numérica	3
1.1.2. Optimización Combinatoria	5
1.2. Vecindad y óptimos locales	6
1.3. Problemas y Algoritmos Eficientes	7
1.4. Complejidad Computacional	10
1.5. Métodos de Búsqueda	18
1.6. Evaluación de metaheurísticas	27
 2. Metaheurísticas aplicadas a problemas de optimización duros	 31
2.1. Introducción	31
2.2. Metaheurísticas	31
2.3. Simulated Annealing	33
2.3.1. Introducción	33
2.3.2. Simulated Annealing, ¿cómo trabaja?	37
2.3.3. Aplicaciones de SA	40
2.4. Tabú Search	40
2.4.1. Introducción	40
2.4.2. TS aplicada al Problema del Viajante de Comercio	41
2.4.3. Extensiones de Tabu Search	47
2.4.4. Aplicaciones de Tabú Search	48
2.5. Algoritmos Evolutivos	49

2.5.1.	Introducción	49
2.5.2.	Descripción General	51
2.5.3.	Estrategias Evolutivas	53
2.5.4.	Programación Evolutiva	55
2.5.5.	Algoritmos Genéticos	57
2.5.6.	Programación Genética	62
2.5.7.	AEs: algunos conceptos importantes	65
2.5.8.	Aplicaciones de AEs	74
3.	Metaheurística ACO	77
3.1.	Optimización en Base a Colonias de Hormigas (ACO) . .	77
3.1.1.	Introducción	78
3.1.2.	La metaheurística ACO	79
3.1.3.	Un algoritmo ACO para el Problema del Viajante de Comercio	85
3.1.4.	Aplicaciones de algoritmos ACO	95
3.2.	Acerca del Teorema NFL (No-Free Lunch)	96
4.	Metaheurísticas y manejo de restricciones	99
4.1.	Introducción	99
4.2.	Acerca de las metaheurísticas	99
4.3.	Manejo de restricciones	109
4.3.1.	Manejo de restricciones en AEs	111
4.3.2.	Técnicas de manejo de restricciones en algoritmos ACO	131
5.	Algoritmos basados en el enfoque de frontera	137
5.1.	Introducción	137
5.2.	Importancia de la búsqueda en la frontera	139
5.3.	Desarrollos de investigación en búsqueda en la frontera .	143
5.4.	Enfoque propuesto	147
5.4.1.	Tipos de problemas considerados	148

5.4.2.	El enfoque de frontera	156
5.5.	Enfoque de frontera en AEs	156
5.6.	Enfoque de frontera en ACO	161
5.7.	Problemas estudiados	174
5.7.1.	Variables Continuas	174
5.7.2.	Variables Discretas	184
6.	Ant Colony Optimization y operadores de frontera	189
6.1.	Introducción	189
6.2.	Problemas continuos (aplicación de SH_C)	192
6.2.1.	Análisis global de SH_C	205
6.3.	Problemas discretos (aplicación de SH_D)	207
6.3.1.	SH_D para MKP	208
6.3.2.	SH_D para SCP	211
6.3.3.	SH_D para MISP	212
6.3.4.	Experimentos y resultados	214
6.3.5.	Análisis global acerca de SH_D	233
7.	Algoritmos Evolutivos y Operadores de Frontera	241
7.1.	Introducción	241
7.2.	Problemas Continuos (aplicación de AE_C)	243
7.2.1.	Análisis global de AE_C	250
7.3.	Problemas discretos (aplicación de AE_D)	252
7.3.1.	AE_D para MKP	253
7.3.2.	AE_D para SCP	257
7.3.3.	AE_D para MISP	262
7.3.4.	Experimentos y resultados	266
7.3.5.	Análisis global de AE_D	272
8.	Discusión General	277
8.1.	Introducción	277
8.2.	Convergencia en SH_C y AE_C	278

8.3. Operadores de frontera <i>ad hoc</i> en AE_C	279
8.4. Parámetros α y β en SH_D	282
8.5. Parámetro ρ en SH_D	284
8.6. Rastro de feromona y exploración del espacio de búsqueda	286
8.7. Observaciones generales	290
9. Conclusiones y Trabajos Futuros	293

Indice de figuras

1.1. Espacio de búsqueda y las respectivas regiones factibles y no factibles.	3
1.2. Algunos ejemplos de reducción a partir de problemas \mathcal{NP} -completos.	17
1.3. Trayectoria descrita por un algoritmo de búsqueda local acorde a una vecindad establecida. Es este caso se puede observar que la vecindad $N(x)$ incluye a todos los puntos que se encuentran a una distancia fija d de cada punto visitado x	20
2.1. Algoritmo general basado en umbrales.	34
2.2. Una implementación general para SA	37
2.3. Estructura de datos tabú que representa los movimientos tabú para TS.	42
2.4. Estructura Tabu - Iteración 0	43
2.5. Estructura Tabú - Iteración 1	44
2.6. Estructura tabú - Iteración 2	45
2.7. Estructura Tabú - Iteración 3	45
2.8. Estructura Tabú - Iteración 4	46
2.9. Una implementación particular de TS para TSP.	47
2.10. Taxonomía histórica común de la computación evolutiva.	51
2.11. Descripción general de un Algoritmo Evolutivo.	52
2.12. El conjunto de apareamiento como un paso previo a la aplicación de los operadores genéticos.	61

2.13. Crossover para representación de árboles en PG. Un subárbol de cada padre (resaltados en gris) es elegido en forma aleatoria e intercambiado entre los padres para crear dos hijos distintos. Este operador automáticamente preserva la sintaxis de la representación de árbol sin necesidad de un proceso de reparación posterior.	64
3.1. Puente con dos brazos para la realización del experimento.	78
3.2. La construcción de una solución implica recorrer el grafo desde un nodo <i>origen</i> hasta alcanzar un nodo <i>destino</i> . Aquellas hormigas que sigan la línea llena, llegarán primero al nodo destino y por ende dirigirán la búsqueda hacia este paso cuando vuelvan hacia el nodo origen.	80
3.3. El SH para TSP. Este algoritmo pertenece a la clase de algoritmos ACO.	87
4.1. Búsqueda iterativa estocástica generalizada	100
4.2. Ideas que subyacen al proceso de generación de nuevas soluciones	102
4.3. Instancia de TSP con 4 ciudades	103
4.4. El grafo muestra la fortaleza del rastro de feromona sobre las conexiones.	107
4.5. Espacio de búsqueda desde la perspectiva de un algoritmo ACO para una instancia de TSP donde la ciudad 1 es la inicial.	108
4.6. Un espacio de búsqueda hipotético con regiones factibles y no factibles.	110
4.7. El problema se divide en dos subespacios y se trata de alcanzar desde cada uno de ellos el punto de <i>ensilladura</i> . .	129
5.1. Hipercubo representando un espacio de búsqueda hipotético formado por cadenas binarias	140

5.2. Oscilación estratégica entre la frontera del espacio factible y no factible.	142
5.3. Operadores genéticos que dan como resultado hijos en la frontera.	144
5.4. Espacio de búsqueda factible determinado por 3 desigualdades.	149
5.5. Con una sencilla búsqueda binaria se llega al punto de corte sobre la frontera.	152
5.6. Algoritmo que permite determinar el punto sobre la frontera dados dos puntos, uno factible y uno no factible. . .	153
5.7. Crossover aritmético aplicado sobre dos pares de puntos para generar un nuevo punto sobre la frontera.	154
5.8. Una variación sobre uno o ambos puntos del par, permite cambiar el punto sobre la frontera.	155
5.9. Visualización del operador de crossover de frontera . . .	159
5.10. Estructura general del operador de crossover de frontera.	160
5.11. Estructura general de la mutación de frontera.	162
5.12. Nido con cuatro puntos	163
5.13. Estructura general de un algoritmo ACO para problemas continuos.	164
5.14. Puntos generados en varios pasos respetando su vecindario respectivo sobre cada dimensión.	165
5.15. Una secuencia que representa una solución parcial \tilde{s}^k en el paso j durante un ciclo particular.	170
5.16. Un conjunto que representa una solución parcial \tilde{S}^k en el paso j durante un ciclo particular.	171
5.17. El Sistema de Hormigas general para problemas de subconjunto.	173
5.18. Función de Keane para dos variables.	176
5.19. Problema G6. Función de Floudas-Pardalos.	179

5.20. Posición aproximada del mejor valor conocido en el espacio de búsqueda factible determinado por las restricciones g_1 y g_2 .	182
5.21. El mejor punto conocido está sobre el espacio de búsqueda determinado por la restricción de igualdad h_1	183
5.22. Matriz que representa los recursos.	185
5.23. Ejemplo de una instancia de SCP (la matriz binaria).	186
5.24. Instancia de MISP de tamaño 10.	187
6.1. Un problema hipotético con tres restricciones ($g_i(x)$ con $i = 1, 2, 3$). La restricción g_2 no está activa en el óptimo (x^*), por lo tanto no es relevante para ser considerada en el proceso de búsqueda.	194
6.2. Convergencia de los mejores valores en cada generación. Se observa una rápida convergencia del algoritmo a valores muy cercanos al óptimo	198
6.3. Instancia de MISP	213
6.4. La figura muestra el promedio del error porcentual según Prom(MVE) para todas las instancias agrupadas por densidad.	224
6.5. Curva de convergencia para distintos valores de β .	225
6.6. Algoritmo para la generación de grafos aleatorios el cual pre-selecciona un conjunto independiente de tamaño k	226
6.7. Un grafo con $n = 10$ nodos donde $V^* = \{1, 3, 5, 6, 8, 10\}$ (indicado con líneas punteadas).	227
7.1. La figura muestra el promedio del error porcentual según Prom(MVE) para todas las instancias agrupadas por densidad.	269
8.1. Convergencia de AE_C y SH_C para los problemas G01, G02279	

8.2.	Convergencia de AE_C para el problema G02 usando un operador de frontera <i>ad hoc</i> y el propuesto en esta tesis. La gráfica de la izquierda muestra el avance hasta la iteración 1000 y la de la derecha hasta la 5000.	280
8.3.	Convergencia de AE_C para los problemas G03, G16 y G17 usando un operador de frontera <i>ad hoc</i> y el generalizado. Las mayores diferencias de velocidad de convergencia se aprecian en los problemas G16 y G17.	282
8.4.	Curva de convergencia para una instancia de MKP para distintos valores de α y β	283
8.5.	Curva de convergencia para una instancia de MKP para distintos valores de la persistencia del rastro ρ con $\alpha = 1$ y $\beta = 5$	285
8.6.	Acumulación del rastro de feromona a través de los ciclos sobre las componentes de una instancia de MKP con 100 variables. Se muestra una instantánea de la acumulación del rastro de feromona cada 100 ciclos.	286

Indice de Tablas

1.1. Relación entre la complejidad de los algoritmos y el tamaño del problema	8
1.2. Los algoritmos polinomiales toman ventaja de los avances tecnológicos	9
5.1. Diferentes <i>regiones</i> en el espacio de búsqueda y las fronteras asociadas.	141
6.1. Valor de penalización usado para G01 es $\mu(t) = 1000$ para $t = 0, 1, \dots, t_{max}$	195
6.2. Aquí se muestran los valores encontrados para la función de Keane con $n = 20, 50$ y 100 variables. Los mejores valores conocidos para cada uno de ellos son $0,803619$ [83, 107], $0,831937$ [157] y ND respectivamente, donde ND indica resultado no disponible.	196
6.3. G03 tiene una restricción $h(x)$ y no es necesario asignar valores a μ . Se muestran los resultados para $n = 20, 50$ variables	197
6.4. Valor de penalización usado para G04 es $\mu(t) = 800000$ para $t = 0, 1, \dots, t_{max}$	199
6.5. Valor de penalización usado para G05 es $\mu(t) = 10$ para $t = 0, 1, \dots, t_{max}$	199
6.6. Valor de penalización usado para G06 es $\mu(t) = 10000$ para $t = 0, 1, \dots, t_{max}$	200

6.7. Valor de penalización usado para G07 es $\mu(t) = 20000$ para $t = 0, 1, \dots, t_{max}$	201
6.8. Valor de penalización usado para G09 es $\mu(t) = 2000$ para $t = 0, 1, \dots, t_{max}$	201
6.9. Valor de penalización usado para G10 es $\mu(t) = 1,05 * \mu(t - 1)$ para $t = 0, 1, \dots, t_{max}$, con $\mu(0) = 200000$	202
6.10. Para G11 no es necesario usar penalización dado que contiene una única restricción por igualdad.	203
6.11. Valor de penalización usado para G13 es $\mu(t) = 0,2$ para $t = 0, 1, \dots, t_{max}$	203
6.12. Valor de penalización usado para G14 es $\mu(t) = 100$ para $t = 0, 1, \dots, t_{max}$	204
6.13. Para G15, G16 y G17 no es necesario usar penalización dado que contienen una única restricción por igualdad. . . .	204
6.14. <i>Comparación SH_C con los resultados de RY y HS</i>	207
6.15. <i>Resultados para 11 instancias de MKP</i>	216
6.16. <i>Resultados para 11 instancias de MKP (continuación)</i> . .	216
6.17. <i>Resultados de SH_D para instancias más difíciles de MKP</i>	218
6.18. Instancias de SCP consideradas en el estudio (primer grupo)	219
6.19. Instancias adicionales de SCP (segundo grupo)	220
6.20. Resultados preliminares obtenidos para algunas instancias de SCP	220
6.21. Resultados para las instancias de SCP tipo 4* y 6* con un incremento en el número de hormigas.	221
6.22. Desempeño de SH_D sobre instancias más difíciles de SCP. Se incluyen la comparación con GENEYS y los valores reportados por P.Chu. [32]. ND: No Disponible	236
6.23. Resultados de SH_D aplicado a diferentes grafos (primer grupo de instancias de MISP)	237
6.24. Algunos valores para las variables X_k en grafos aleatorios	237
6.25. Algunos valores para las variables X_k en grafos aleatorios	237

6.26. Resultados de SH_D aplicado a diferentes grafos aleatorios (segundo grupo de instancias de MISP)	238
6.27. Resultados de SH_D (tercer grupo de instancias de MISP - DIMACS).	239
7.1. Valor de penalización usado para G01 es $\mu(t) = 500$ para $t = 0, 1, \dots, t_{max}$	244
7.2. Aquí se muestran los valores encontrados para la función de Keane con $n = 20, 50$ y 100 variables. Los mejores valores conocidos para cada uno de ellos son $0,803619$ [83, $107]$, $0,831937$ [157] y ND respectivamente, donde ND indica resultado no disponible.	245
7.3. G03 tiene una restricción $h(x)$ y no es necesario asignar valores a μ . Se muestran los resultados para $n = 20, 50$ variables	245
7.4. Valor de penalización usado para G04 es $\mu(t) = 8000000$ para $t = 0, 1, \dots, t_{max}$	246
7.5. Valor de penalización usado para G05 es $\mu(t) = 10$ para $t = 0, 1, \dots, t_{max}$	246
7.6. Valor de penalización usado para G06 es $\mu(t) = 10000$ para $t = 0, 1, \dots, t_{max}$	247
7.7. Valor de penalización usado para G07 es $\mu(t) = 10$ para $t = 0, 1, \dots, t_{max}$	247
7.8. Valor de penalización usado para G09 es $\mu(t) = 10$ para $t = 0, 1, \dots, t_{max}$	248
7.9. Valor de penalización usado para G10 es $\mu(t) = 1,05 * \mu(t - 1)$ y $\mu(0) = 200000$ para $t = 0, 1, \dots, t_{max}$	249
7.10. G11 tiene una restricción $h(x)$ y no es necesario asignar valores a μ	249
7.11. Valor de penalización usado para G13 es $\mu(t) = 0,1$ para $t = 0, 1, \dots, t_{max}$	249

7.12. Valor de penalización usado para G14 es $\mu(t) = 1$ para $t = 0, 1, \dots, t_{max}$	250
7.13. G15, G16 y G17 tienen una restricción $h(x)$ y no es nece- sario asignar valores a μ	251
7.14. <i>Comparación de AE_C con los resultados de RY y HS</i> . .	252
7.15. Resultados para las instancias for^* de MKP obtenidas de la OR Library	267
7.16. Resultados para las instancias $(5+10).100^*$ de MKP ob- tenidas de la OR Library y generadas por Paul Chu[32] .	268
7.17. Resultados del AE_D para un conjunto de instancias de SCP	274
7.18. Resultados para el conjunto de instancias de MISP gene- radas según los métodos propuestos por Khuri et al. [12]	275
7.19. Resultados para el conjunto de instancias MISP generadas según el método descrito por Bollobas et al. [25]	275
7.20. Resultados del AE_D para las instancias de DIMACS . . .	276

Capítulo 1

Introducción: Conceptos y Definiciones

Este capítulo introduce algunos conceptos relacionados a problemas de optimización numérica y combinatorios, vecindad y búsqueda local. Una breve introducción a *\mathcal{NP} -Complejidad* es también presentada. El capítulo concluye con una descripción de varios enfoques de búsqueda y optimización aplicados a problemas *\mathcal{NP} -Complejos* entre los que se incluyen a las metaheurísticas.

1.1. Problemas de optimización

Muchos problemas de índole práctica o teórica están caracterizados por la elección de la “mejor” configuración o conjunto de parámetros para alcanzar un determinado objetivo [124].

Los problemas de optimización pueden dividirse en dos principales categorías: (1) aquellos que involucran variables continuas (problemas de optimización numérica) y (2) aquellos que involucran variables discretas (problemas de optimización combinatoria).

La solución a un problema continuo involucra un conjunto de variables reales, mientras que en el caso discreto, la solución consta de un objeto de un conjunto finito o contablemente infinito (por ejemplo, el conjunto de números enteros), permutaciones o grafos. Estas dos clases de problemas

generalmente tienen diferentes características de la misma forma que se diferencian las respectivas técnicas para resolverlos. Cualquiera sea el caso, el método de optimización a ser usado tiene que tener en cuenta las características del espacio de búsqueda \mathcal{S} — el conjunto de todas las posibles soluciones para dicho problema.

A su vez, muchos problemas de importancia práctica involucran diferentes clases de restricciones las que pueden tornar más difícil encontrar un método que pueda resolver el problema en forma aceptable. En este contexto la palabra aceptable implica poder encontrar rápidamente una solución cuyo valor objetivo sea muy cercano al óptimo.

La incorporación de restricciones a la formulación de un problema de optimización, implica una división del espacio de búsqueda \mathcal{S} en dos regiones perfectamente diferenciadas, la región de soluciones factibles (\mathcal{F}) y la región de soluciones no factibles ($\mathcal{U} = \mathcal{S} - \mathcal{F}$), esto es, el conjunto de soluciones que satisfacen y no satisfacen a las restricciones del problema respectivamente. Las características de las restricciones determinarán las formas del conjunto \mathcal{F} y por ende la dificultad del problema.

De esta manera, podemos definir un problema de optimización como sigue: Dado un espacio de búsqueda \mathcal{S} , el conjunto de soluciones factibles $\mathcal{F} \in \mathcal{S}$ y una función objetivo f , se necesita encontrar un elemento $x \in \mathcal{F}$ tal que $f(x) \leq f(y)$ ¹, para todo $y \in \mathcal{F}$. La definición anterior es lo suficientemente general para ser aplicada a diferentes tipos de problemas. Sin embargo, para un determinado espacio de búsqueda es necesario conocer la función objetivo (f), la representación para las soluciones y la manera en que dichas soluciones son generadas durante el proceso de búsqueda.

El punto $x \in \mathcal{F}$ que satisfaga esta definición es llamado una solución global. Sin embargo, encontrar tal solución global para un problema podría ser extremadamente difícil y muchas veces encontrar la mejor

¹Notar que \leq implica un problema de minimización.

solución es más fácil si sólo nos concentramos en un subconjunto relativamente pequeño (factible y tal vez no factible) del conjunto conformado por todas las posibles soluciones. Es decir que mientras menos alternativas existan, es decir, reducir el espacio de potenciales soluciones a considerar, el proceso de búsqueda puede ser más efectivo. Esta es una observación fundamental que subyace en muchas técnicas de optimización [118].

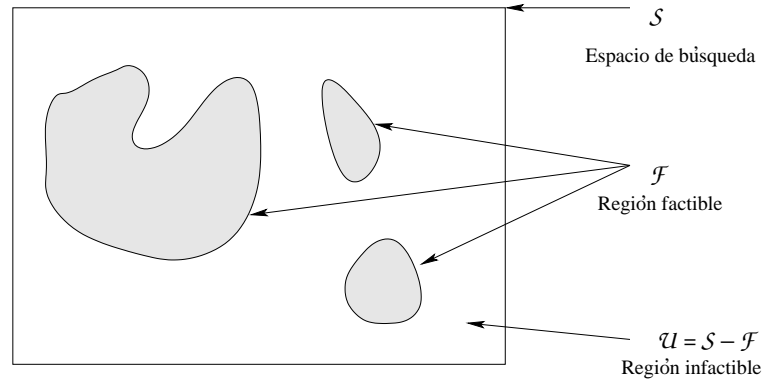


Figura 1.1: Espacio de búsqueda y las respectivas regiones factibles y no factibles.

1.1.1. Optimización Numérica

El problema general de programación no lineal (de aquí en más referido como NLP²) puede ser formulado como el objetivo de encontrar un punto \mathbf{x} de manera tal que:

$$\text{optimice } f(\mathbf{x}) \quad \mathbf{x} = (x_1, x_2, \dots, x_n) \in \mathbb{R}^n \quad (1.1)$$

donde $\mathbf{x} \in \mathcal{F} \subset \mathcal{S}$ (Fig. 1.1). El conjunto $\mathcal{S} \in \mathbb{R}$ define el espacio de búsqueda y el conjunto $\mathcal{F} \subseteq \mathcal{S}$ define el espacio de búsqueda *factible*. El espacio de búsqueda \mathcal{S} es definido como un rectángulo n -dimensional \mathbb{R}^n (dominios de las variables definidas según sus límites superiores e inferiores):

$$l(i) \leq x_i \leq u(i) \quad 1 \leq i \leq n$$

²Siglas en Inglés para Non Linear Programming

mientras que el conjunto factible \mathcal{F} es definido por la intersección de \mathcal{S} y el conjunto de $m \geq 0$ restricciones adicionales del problema:

$$g_j \leq 0 \text{ para } j = 1, \dots, q \quad h_j = 0 \text{ para } j = q + 1, \dots, m$$

Dado cualquier punto $\mathbf{x} \in \mathcal{F}$, las restricciones g_k que satisfacen $g_k(\mathbf{x}) = 0$ son llamadas las restricciones activas en el punto \mathbf{x} . Por extensión, la restricciones de igualdad h_j son también llamadas activas en todos los punto de \mathcal{S} .

El problema NLP es intratable. Si la función objetivo f y las funciones g'_j s y h'_j s que determinan las restricciones son arbitrarias, luego no existen muchas alternativas más allá de los métodos de búsqueda exhaustiva para resolverlos. Por esta razón, varios casos especiales de NLP han sido identificados. Por ejemplo, si todas las funciones g'_j s y h'_j s son lineales, tales problemas son llamados *problemas de optimización linealmente restringidos*. Además, si la función objetivo f es polinomial y a lo sumo cuadrática, el problema es llamado *problema de programación cuadrática*. El caso mejor conocido de un problema cuadrático es llamado *problema de programación lineal*. Existe también un caso importante llamado *optimización no restringida*, donde no existen restricciones de ningún tipo, es decir, $m = 0$ y $\mathcal{F} = \mathcal{S}$.

Una de las principales dificultades para resolver NLP es el problema de los óptimos locales. Un punto factible $\mathbf{y} \in \mathcal{F}$ es un óptimo local para la función objetivo f si y solamente si existe $\epsilon > 0$ tal que para todo \mathbf{x} en la vecindad $-\epsilon$ de $\mathbf{y} \in \mathcal{F}$, $f(\mathbf{y}) \leq f(\mathbf{x})$ — esto es para problemas de minimización. Existen diferentes condiciones de optimalidad, que si se verifican, nos permiten asegurar que la solución encontrada es un candidato a óptimo global o directamente que dicha solución es un óptimo global. Por ejemplo, en Nocedal et al. [91] se describen, entre muchas otras, las condiciones de necesidad y suficiencia de Kuhn-Tucker las cuales involucran multiplicadores de Lagrange y las derivadas de las restricciones del problema. Sin embargo, para poder aplicar cualquier

tipo de condición de optimalidad, el requisito indispensable es que la función objetivo y las restricciones sean diferenciables. Si la diferenciabilidad es posible, entonces hay condiciones que permiten verificar si una solución es un candidato al óptimo global, aunque se requieren condiciones bastante fuertes para poder demostrar que éste es el óptimo global.

Debido a que resulta bastante difícil poder verificar las condiciones más fuertes (suficiencia), entonces suele decirse que, en el mejor de los casos, la solución que se encontró es un óptimo local (cuando las condiciones de necesidad son aplicables). Sin embargo, es evidente que el problema general de optimización no lineal es un problema abierto. Basta con que haya una restricción de igualdad no lineal para que no apliquen las condiciones más fuertes y no pueda demostrarse optimalidad global. Esta situación es precisamente lo que justifica el uso de metaheurísticas como método alternativo para su resolución.

1.1.2. Optimización Combinatoria

Un problema de optimización combinatoria es especificado a través de un conjunto de instancias del problema y puede ser un problema de maximización o minimización. Una instancia de un problema de optimización combinatoria es un par (\mathcal{F}, f) , donde el conjunto de posibles soluciones es $\mathcal{F} \subseteq S$, es decir, el conjunto de soluciones factibles, S es el espacio de búsqueda y la función objetivo (costo) f es una transformación $f : \mathcal{F} \rightarrow R$. El problema³ es encontrar una solución globalmente óptima, esto es, un $i^* \in \mathcal{F}$ tal que $f(i^*) \leq f(i)$ para todo $i \in \mathcal{F}$. Más aún, $f^* = f(i^*)$ denota el costo óptimo, y $\mathcal{F}^* = \{i \in \mathcal{F} | f(i) = f^*\}$ denota el conjunto de soluciones óptimas.

La instancia (\mathcal{F}, f) no es en general dada en forma explícita, es decir, dando un listado de todas las soluciones y sus respectivos costos (o valores objetivos en general). Por el contrario, la que se tiene es una representación compacta de una instancia y algoritmos polinomiales pa-

³Aquí consideramos un problema de minimización.

ra verificar si una solución pertenece a \mathcal{F} y para computar el costo de cualquier solución en \mathcal{F} . El tamaño de esta representación, es decir, el número de bits necesarios para almacenarlo en una computadora es frecuentemente tomado como el tamaño de la instancia del problema [128]. El conjunto solución es frecuentemente representado por un conjunto de *variables de decisión* cuyos valores están dentro de un rango específico. Luego, una solución es representada por asignación de valores a dichas variables las cuales pueden estar directamente relacionadas al modelo que es usado para formular el problema. Es importante destacar que la representación de las soluciones es una cuestión clave en muchos métodos heurísticos. Por un lado, determina las características del espacio de búsqueda y además tiene una influencia directa en el diseño de los operadores que permiten explorar dicho espacio, por ejemplo, operadores de mutación, crossover o recombinación, búsqueda local, heurísticas greedy, etc.

1.2. Vecindad y óptimos locales

Dado un punto factible $s \in \mathcal{F}$ para un problema particular, es importante en muchas situaciones definir el conjunto $\mathcal{N}(s)$ de puntos que están “cercaños” a s . Así, para un problema de optimización con instancias (\mathcal{F}, f) , la vecindad es una transformación $\mathcal{N} : \mathcal{F} \rightarrow 2^{\mathcal{F}}$ definido para toda instancia. Si $\mathcal{F} = \mathbb{R}$, el conjunto de puntos dentro de una distancia Euclidiana fija provee una vecindad natural. En muchos problemas combinatorios, la elección de \mathcal{N} depende críticamente de la estructura \mathcal{F} . Muchos métodos de búsqueda están basados en estadísticas de la vecindad alrededor de un punto dado; esto significa que la secuencia de puntos que esas técnicas generan mientras buscan la mejor solución posible, se basan en información *local* en cada paso del camino que recorre. Estas técnicas están diseñadas para ubicar soluciones dentro de una vecindad del punto actual que tienen mejores valores de la función objetivo. Este

tipo de técnicas son conocidas como estrategias de búsqueda “local” o “en la vecindad”.

Métodos de búsqueda local presentan un interesante balance entre el tamaño de la vecindad $\mathcal{N}(x)$ y la eficiencia de la búsqueda. Si el tamaño es relativamente pequeño, el algoritmo será capaz de cubrir rápidamente el total de la vecindad. De esta manera, sólo unas pocas soluciones deber ser evaluadas antes de decidir qué solución debería ser considerada en un próximo paso. La desventaja de tener una vecindad muy pequeña está directamente relacionada con un incremento en la probabilidad de quedar atrapado en un óptimo local. Esta situación sugiere el uso de vecindades de mayor tamaño, es decir, un mayor rango de visibilidad podría ayudar a tomar mejores decisiones. En particular, si la visibilidad fuera irrestricta (en el caso extremo el tamaño de la vecindad sería tan grande como el espacio de búsqueda), eventualmente, se podría encontrar la serie de pasos a seguir hasta encontrar el punto óptimo. Sin embargo, el número de evaluaciones se tornaría inmanejable, llegando al extremo de requerir un tiempo inadmisiblemente tan prolongado como el tiempo de vida del Universo. Por consiguiente, cuando se usan métodos de búsqueda local, el tamaño apropiado de la vecindad no puede ser determinado arbitrariamente. Por el contrario, debería ajustarse cuidadosamente al problema en particular.

1.3. Problemas y Algoritmos Eficientes

¿Cuándo un problema computacional debería ser considerado satisfactoriamente resuelto? Una posible respuesta podría estar relacionada a la performance de los algoritmos conocidos para ese problema. En la actualidad, existe una aceptación generalizada en el ámbito de Ciencias de la Computación de que un algoritmo es una solución de utilidad práctica, solamente si su complejidad crece *polinomialmente* con respecto al tamaño de la entrada. En consecuencia, algoritmos de complejidad

$\mathcal{O}(n)$ u $\mathcal{O}(n^3)$ podrían ser aceptables. Naturalmente, los algoritmos para los cuales la complejidad asintótica no es polinomial en sí misma, pero están acotados por un polinomio, también son considerados dentro de esta clase.

Ejemplos de estos últimos son $n^{2,5}$ y $n \log n$. Por el contrario, crecimientos *exponenciales* como 2^n y aún peores como $n!$ son considerados más ineficientes, a medida que el tamaño de problema crece (ver Tabla 1.1). Los problemas que sólo aceptan esta clase de algoritmos son llamados *intratables*.

Tabla 1.1: Relación entre la complejidad de los algoritmos y el tamaño del problema

Complejidad Temporal	Valores aproximados		
n	10	100	1000
$n \log n$	33	664	9966
n^3	1000	1,000,000	10^9
$10^6 n^8$	10^{14}	10^{22}	10^{30}
2^n	1024	$1,27 \times 10^{30}$	1.05×10^{301}
$n^{\log n}$	2099	$1,93 \times 10^{13}$	7.89×10^{29}
$n!$	3,628,800	10^{158}	4×10^{2567}

Otra característica positiva de los algoritmos polinomiales es que, en algún sentido, toman ventaja de los avances tecnológicos⁴. Por ejemplo, si la velocidad de las computadoras es multiplicado por 10, el tamaño de la instancia más grande que puede ser resuelta por un algoritmo de tiempo polinomial en una hora, será multiplicado por una constante entre 1 y 10. Por otro lado, un algoritmo exponencial experimentará solamente un incremento *aditivo* en el tamaño de la instancia que pueda ser resuelto en el mismo periodo de tiempo [128]. La forma de obtención de dichas cifras es como sigue. Si por ejemplo un algoritmo tiene complejidad 2^n y el tamaño de la instancia que resuelve en una hora es $n = 40$, para una computadora 10 veces más rápida, sería equivalente a calcular el tamaño

⁴Con respecto a este tema es importante destacar que debido al importante avance tecnológico de los últimos años, ha ocurrido una explosión en el desarrollo e investigación de distintas metaheurísticas.

de la instancia que podría resolver la misma computadora en un tiempo 10 veces mayor que el necesario para resolver la instancia de tamaño 40. Esto es, en términos temporales, 10×2^{40} . Dado que estamos interesados en el tamaño de la instancia que resolvería en ese “tiempo”, aplicamos \log_2 a la expresión 10×2^{40} , lo que da como resultado $\log_2(10 \times 2^{40})$ o bien, $\log_2(10) + \log_2(2^{40}) = 3,32 + 40$. Eliminando los decimales se obtiene el valor especificado en la tabla 1.1, fila 5. Los demás valores de la tabla se obtienen de una manera similar.

Tabla 1.2: Los algoritmos polinomiales toman ventaja de los avances tecnológicos

Complejidad Temporal	Tamaño de la instancia resuelta en una hora	Tamaño de la instancia resuelta en una hora con una computadora 10 veces más rápida
n	10^{12}	10^{13}
$n \log n$	0.948×10^{11}	0.87×10^{12}
n^3	10^4	2.15×10^4
$10^8 n^4$	10	18
2^n	40	43
10^n	12	13
$n^{\log n}$	79	95
$n!$	14	15

La dicotomía entre cotas de tiempo polinomiales y no polinomiales, y la identificación de algoritmos acotados por un polinomio con la noción intuitiva de “computación de utilidad práctica” no siempre es muy clara. Existen computaciones eficientes que no son polinomiales y computaciones polinomiales que no son eficientes en la práctica. Por ejemplo, un algoritmo n^{80} podría tener sus limitaciones prácticas y un algoritmo con velocidad de crecimiento exponencial tal como $2^{\frac{n}{100}}$ podría ser más útil. Sin embargo, la experiencia con distintos tipos de algoritmos ha mostrado que las velocidades extremas de crecimiento, tales como n^{80} o $2^{\frac{n}{100}}$, son raras en la práctica. En general, algoritmos polinomiales típicamente tienen pequeños exponentes y constantes multiplicativas razonables, y por

cierto, los algoritmos exponenciales tienen usualmente poca importancia práctica [128, 129].

Es importante destacar que los algoritmos polinomiales tienen interesantes propiedades de clausura: Los algoritmos polinomiales pueden ser combinados para resolver casos especiales del mismo problema o de problemas similares dado que un algoritmo polinomial puede invocar a otro como una subrutina y el algoritmo resultante será polinomial.

1.4. Complejidad Computacional

El conjunto $\mathcal{F} \subset \mathcal{S}$ de soluciones factibles y la función f , presentados en la sección 1.1.2 pueden ser dados en forma implícita por dos algoritmos $A_{\mathcal{F}}$ y A_f respectivamente. El algoritmo $A_{\mathcal{F}}$ decidirá, dados un objeto combinatorio $s \in S$ y un conjunto de parámetros P , si s es un elemento de \mathcal{F} . Por su parte, el algoritmo A_f retornará el valor de $f(s)$, dados una solución factible $s \in \mathcal{F}$ y otro conjunto de parámetros Q .

Así, una *instancia* de un problema combinatorio puede ser definida a través de la representación de los parámetros en P y Q usando un alfabeto finito y un esquema razonable de codificación [128]. Dicha instancia podría representar un grafo, una secuencia de enteros (representando tal vez arreglos o matrices), una familia de conjuntos, etc.

Por ejemplo, para una instancia del Problema de Viajante de Comercio (de ahora en más TSP, sus siglas en Inglés para Travelling Salesperson Problem), el número n de ciudades y las entradas de la matriz de distancias $[d_{ij}]$ podrían representar respectivamente los parámetros P y Q . Para este problema, un algoritmo del tipo $A_{\mathcal{F}}$ recibirá como entrada un objeto s y un entero n . El mismo determinará si s es un tour válido⁵ de n ciudades. En el caso del algoritmo tipo A_f , los datos de entradas son un tour $s \in \mathcal{F}$ y la matriz de distancias $[d_{ij}]$, pero la salida estará representada por el costo $f(s)$ el cual es obtenido sumando las distancias

⁵Un camino Hamiltoniano de n ciudades.

entre las ciudades involucradas en el correspondiente tour válido s .

Una vez determinada la representación de las instancias del problema, en general como una secuencia de símbolos, se define el tamaño de la instancia (muchas veces referido como *tamaño del problema*) como la longitud de la secuencia que la representa, es decir, la cantidad de símbolos que la componen. Por ejemplo, si la instancia de un problema es un grafo, una representación directa para el mismo puede ser una matriz de adyacencia de tamaño $|V| \times |V|$. Sin embargo, esta representación puede no ser muy económica cuando la matriz es dispersa. Alternativamente se podría usar listas de adyacencia para minimizar el espacio ocupado u otro enfoque. De cualquier manera esta situación tiene que ver principalmente con el espacio usado según la representación usada, y no implica necesariamente la disminución del tamaño del problema original.

En general, los algoritmos $A_{\mathcal{F}}$ y A_f tiene una complejidad temporal acotada por un polinomio, lo cual es una premisa conveniente en este contexto, aunque existen excepciones en las cuales se verifica que $f()$ es dura para computar.

Bajo el supuesto de complejidad polinomial para $f()$, un problema de *optimización combinatoria* es uno de los siguientes problemas computacionales [128]:

Versión de Optimización

Dada una representación de los parámetros P y Q para los algoritmos $A_{\mathcal{F}}$ y A_f , respectivamente, encontrar la solución factible óptima.

Versión de Evaluación

Dados P y Q , encontrar el costo de la solución óptima.

Versión de Reconocimiento

Dada una instancia, es decir, una representación de P y Q , y un entero L , ¿existe una solución factible $s \in \mathcal{F}$ tal que $f(s) \leq L$? ⁶.

⁶O bien, $f(s) \geq L$ para un problema de maximización

En consecuencia, la *versión de evaluación* no puede ser más difícil que la *versión de optimización*. La tercera versión de un problema de optimización combinatoria es importante desde el punto de vista del estudio de la complejidad del problema dado que representa el prototipo de los problemas tradicionalmente estudiados en teoría de la computación (decibilidad). Esta versión es de hecho una *pregunta* que puede ser contestada con un *sí* o un *no*. Dado que se compara el costo óptimo $f(s^*)$ con L , la *versión de reconocimiento* no es mucho más difícil que resolver la *versión de evaluación*, la que reporta *sí*, siempre que $f(s^*) \leq L$, asumiendo que la *versión de evaluación* reporta $f(s^*)$. Por lo tanto, cada una de las versiones de optimización, evaluación y reconocimiento, en este orden, no es más difícil que la anterior, asumiendo que f es una función con complejidad polinomialmente acotada.

Tal como fue establecido en la sección 1.3, aceptamos que es razonable entender por “algoritmo eficiente” a aquel algoritmo que requiere un número de pasos que crece polinomialmente con el tamaño de la entrada. Lamentablemente existe un inmenso número de problemas para los cuales no es conocido ningún algoritmo eficiente según la teoría de NP-completitud [128, 68, 129].

A continuación se da una breve introducción a la noción de problemas NP-completos. Previamente introduciremos la definición de Máquina de Turing, un dispositivo abstracto ampliamente conocido y comunmente usado en Teoría de la Computación [92, 128, 164].

La Máquinas de Turing son particularmente aptas para resolver cierta clase de problemas relacionados con cadenas, específicamente computación de funciones de cadenas, reconocimiento y decisión de lenguajes.

Una Máquina de Turing es una cuádrupla $N = (K, \Sigma, \Delta, s)$ donde K es el conjunto de estados, Σ es el alfabeto, y s es el estado inicial. Δ es una función de $K \times \Sigma$ a $(K \cup \{h, \text{“si”}, \text{“no”}\}) \times \Sigma \times \{\leftarrow, \rightarrow, -\}$.

Definición 1.1 Sea $L \subset (\Sigma - \{B\})^*$ un lenguaje, es decir, un conjunto de cadenas de símbolos. Sea M una Máquina de Turing tal que, para cada

cadena $x \in (\Sigma - \{B\})^*$, si $x \in L$, luego $M(x) = \text{“sí”}$ (es decir, cuando M es iniciada con la entrada x , se detiene en el estado de aceptación “sí”). Y si $x \notin L$, luego $M(x) = \text{“no”}$. Decimos entonces que M decide L . Si L es decidido por alguna Máquina de Turing M , luego L es llamado un *lenguaje recursivo*. Por otro lado, decimos que M simplemente *acepta* o *reconoce* L siempre y cuando, para cualquier cadena $x \in (\Sigma - \{B\})^*$, si $x \in L$, luego $M(x) = \text{“sí”}$; sin embargo, si $x \notin L$, luego M puede no detenerse y por ende, nunca obtener una respuesta. Esta situación se denota como $M(x) = \nearrow$. Si L es *aceptado* por alguna Máquina de Turing M , luego L es denominado *recursivamente enumerable*.

Es claro que la versión de reconocimiento de un problema de optimización combinatoria es equivalente a decidir a través de una Máquina de Turing el lenguaje recursivo L_P , donde L_P representa el conjunto de cadenas de instancias “sí” del problema P . Sin embargo, los problemas de reconocimiento, y por ende de evaluación y optimización, no tienen la misma complejidad. En lo siguiente definimos algunas clases importantes de complejidad para clasificar los problemas de optimización combinatoria.

Definición 1.2 Supongamos que un lenguaje $L \subset (\Sigma - \{B\})^*$ es decidido por una MT operando en tiempo $f(n)$. Luego, decimos que $L \in TIME(f(n))$. En otras palabras, $TIME(f(n))$ es el conjunto de lenguajes que pueden ser decididos por una MT dentro de un tiempo acotado por $f(n)$.

Definición 1.3 Un lenguaje L es *decidible en tiempo polinomial* si existe una MT estándar que acepta L en tiempo $O(n^r)$ donde r es un número natural independiente de n . La familia de lenguajes decidibles en tiempo polinomial es denotada por \mathcal{P} . Alternativamente es posible expresar la clase \mathcal{P} como:

$$\mathcal{P} = \bigcup_{r>1} TIME(n^r)$$

La eficiencia de un algoritmo es medida por la complejidad temporal de su implementación sobre una MT estándar. Es posible, sin embargo, elegir como modelo de computación MT con múltiples pistas, múltiples cintas, etc., para realizar la evaluación del algoritmo dado que la clase \mathcal{P} es invariante independientemente de la elección de la MT determinística elegida para el análisis. Por ejemplo, un lenguaje aceptado por una MT de múltiples pistas en tiempo $O(n)$ es aceptado también por una MT estándar en tiempo $O(n)$ [164]. Lamentablemente, esta situación no se mantiene cuando el modelo de computación es no determinístico.

Una MT no determinística es una cuádrupla $N = (K, \Sigma, \Delta, s)$, donde K , Σ y s son definidas en forma equivalente a una MT determinística estándar definida previamente. Sin embargo, Δ no se mantiene como una función de $K \times \Sigma$ en $(K \cup \{h, "si", "no"\}) \times \Sigma \times \{\leftarrow, \rightarrow, -\}$, ahora es una relación $\Delta \subset [(K \cup \{h, "si", "no"\}) \times \Sigma \times \{\leftarrow, \rightarrow, -\}]$. Esto significa que para cada combinación de símbolos, más de un movimiento es posible o inclusive, ninguno [129].

Lo que hace a las máquinas no determinísticas diferentes y potentes es la noción muy amplia de lo que significa para este tipo de máquina resolver un problema. Sea L un lenguaje y N una Máquina de Turing no determinística. Decimos que N decide L si para cualquier $x \in \Sigma$, se cumple lo siguiente: $x \in L$ si y solamente si $(s, \triangleright, x) \xrightarrow{N^*} ("si", w, u)$ para algún w y u .

Una MT no determinística N decide el lenguaje L en tiempo $f(n)$, donde $f : N_0 \rightarrow N_0$, si N decide L , y para cualquier $x \in \Sigma^*$, si $(x, \triangleright, x) \xrightarrow{N^k} (Q, u, w)$, luego $k \leq f(|x|)$.

Definición 1.4 Supongamos que un lenguaje $L \subset (\Sigma - \{B\})^*$ es decidido por una MT no determinística operando en tiempo $f(n)$. Luego decimos que $L \in NTIME(f(n))$. Así, $NTIME(f(n))$ es el conjunto de lenguajes que pueden ser decididos por una MT no determinística dentro de un tiempo acotado por $f(n)$.

Definición 1.5 Un lenguaje L es *decidible en tiempo polinomial no determinístico*, si existe una MT no determinística M que acepta L en tiempo $O(n^r)$ donde r es un número natural independiente de n . La familia de lenguajes decidibles en tiempo polinomial no determinístico es denotado \mathcal{NP} . De la misma manera, podemos expresar la clase \mathcal{NP} como:

$$\mathcal{NP} = \bigcup_{r>1} NTIME(n^r)$$

donde n es la longitud de la entrada.

El no determinismo es un tema central en Teoría de Complejidad dada su vinculación con *aplicaciones* relacionadas con lógica, optimización combinatoria e inteligencia artificial.

Por otro lado, un problema típico en inteligencia artificial y optimización combinatoria es la búsqueda de una solución de menor costo que satisfaga ciertas restricciones entre una cantidad exponencial de posibles soluciones. En este sentido, una MT no determinística puede buscar una solución en un espacio de tamaño exponencial sin demasiado esfuerzo.

Existen varios modelos de computación que pueden ser simulados entre ellos con una pérdida polinomial en la eficiencia [129]. Sin embargo, una MT no determinística que no es un modelo razonable de computación, puede ser simulado por modelos realísticos de computación pero con una pérdida exponencial en la eficiencia. El hecho que esta pérdida exponencial sea inherente o es un problema de nuestro entendimiento limitado del no determinismo es conocido como el problema $\mathcal{P} \stackrel{?}{=} \mathcal{NP}$, el cual es un problema abierto.

En la actualidad, sólo conocemos métodos exponenciales para adaptar un algoritmo no determinístico a su correspondiente versión determinística. Esto es expresado a través del siguiente Teorema [129]: Supongamos que el lenguaje L es decidido por una MT no determinística N en tiempo $f(n)$, luego L es decidido por una MT M de 3-cintas en tiempo $O(c^{f(n)})$, donde $c > 1$ es una constante que depende de n .

Este teorema puede ser expresado, usando notación de clases de complejidad, como sigue:

$$NTIME(f(n)) \subseteq \bigcup_{c>1} TIME(c^{f(n)}).$$

A fin de caracterizar a aquellos problemas que son inherentemente difíciles, es importante además introducir el concepto de reducción. La reducción es un concepto ampliamente usado para demostrar que una importante cantidad de problemas de Teoría de Computación son no decidibles. Similarmente, a través de la reducción, podemos relacionar muchos problemas de optimización acorde a su inherente complejidad.

Definición 1.6 Sean Q y L lenguajes definidos sobre alfabetos Σ_1 y Σ_2 , respectivamente. Decimos que Q es reducible a L en tiempo polinomial, si existe una función $f : \Sigma_1^* \rightarrow \Sigma_2^*$ con $f \in TIME(n)$, tal que $u \in Q$, si y solamente si, $f(u) \in L$.

Ha sido mostrado también [68, 129, 127] que si un lenguaje Q es reducible a un lenguaje $L \in \mathcal{P}$ en tiempo polinomial, luego $Q \in \mathcal{P}$. Sin embargo, para muchos problemas del mundo real, no existe una solución en \mathcal{P} o una reducción a un problema (versión de reconocimiento) en \mathcal{P} . Sin embargo, a veces es más fácil mostrar que dichos problemas pertenecen a la clase \mathcal{NP} .

Definición 1.7 Un lenguaje L es llamado \mathcal{NP} -duro si para cada $Q \in \mathcal{NP}$, Q es reducible a L en tiempo polinomial. Un lenguaje \mathcal{NP} -duro que está en \mathcal{NP} es denominado \mathcal{NP} -completo.

Un lenguaje \mathcal{NP} -completo puede ser considerado como un lenguaje universal en la clase \mathcal{NP} o bien que es un lenguaje tan difícil como cualquiera de los lenguajes de dicha clase. El descubrimiento de una máquina⁷ que acepte un lenguaje \mathcal{NP} -completo en tiempo polinomial

⁷Se asume una Máquina de Turing como el representante más ampliamente difundido de Modelo de Computación.

determinístico puede ser usado para construir máquinas que acepten todos los lenguajes en \mathcal{NP} en tiempo polinomial determinístico. En consecuencia se podría dar un respuesta afirmativa a la famosa pregunta $\mathcal{P} = \mathcal{NP}$?

Históricamente, el problema de satisfactibilidad fue le primer problema de decisión para el cual se mostró que era \mathcal{NP} -completo (Teorema de Cook [128, 129, 68]).

A partir de aquí, fue mostrado que una importante cantidad de problemas eran \mathcal{NP} -completos usando el concepto de reducción previamente descrito — SAT, 3-SAT, Cobertura de vértices, Max-Clique, Circuito Hamiltoniano, El Viajante de Comercio, Conjunto Independiente Máximo, Problema de Mochilas Múltiples⁸ y muchos otros [68]. La figura 1.2 muestra algunas posible reducciones a partir de SAT. La prueba de

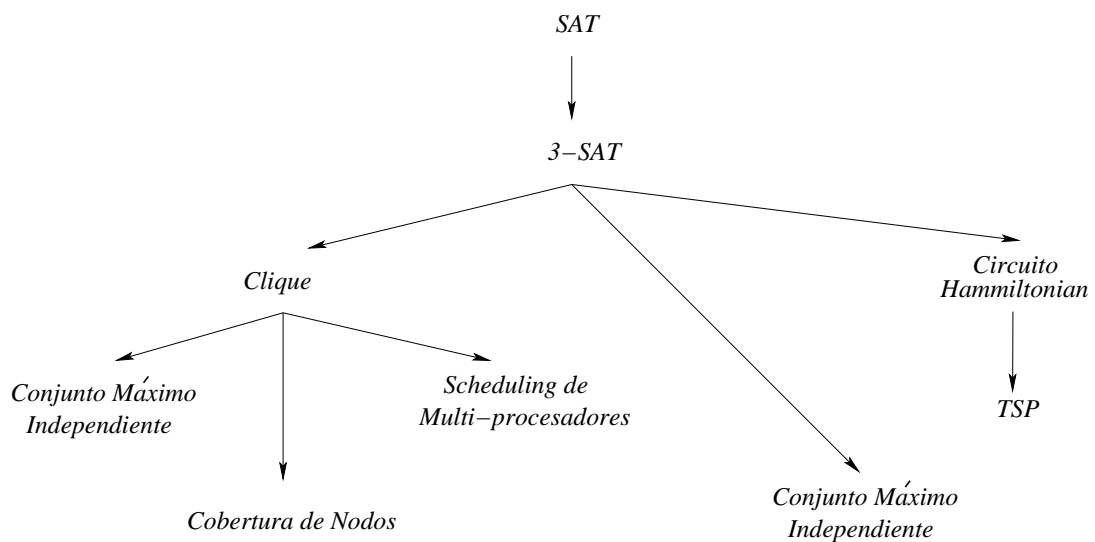


Figura 1.2: Algunos ejemplos de reducción a partir de problemas \mathcal{NP} -completos.

\mathcal{NP} -completitud es típicamente el primer paso en el análisis de problemas computacionales según los métodos de Teoría de algoritmos y complejidad. Una vez que se ha mostrado que un problema es \mathcal{NP} -completo, estamos motivados a explorar posibilidades menos ambiciosas que resol-

⁸De aquí en más usaremos a lo largo de este documento, las respectivas siglas en Inglés para la mayoría de los problemas considerados o referenciados: Multiple Knapsack Problem (MKP), Set Covering Problem (SCP) y Maximum Independent Set Problem (MISP).

ver el problema exactamente y eficientemente para cualquier instancia del mismo. Este hecho ha llevado a muchos investigadores a rediseñar sus estrategias para atacar esta clase de problemas intratables. Hay varias alternativas que son menos ambiciosas que intentar resolver problemas de optimización \mathcal{NP} -*completos* en forma exacta y eficientemente, pero que sin embargo, pueden dar soluciones aceptables y muchas veces las óptimas dependiendo del método, problemas e instancias del problema. Y dentro de las instancias, la calidad de la solución encontrada depende en muchos casos del tamaño y tipo de la instancia. Algunas de esas alternativas para realizar la búsqueda en un espacio de soluciones son descriptas brevemente a continuación.

1.5. Métodos de Búsqueda

- Búsqueda exhaustiva: Tal como el nombre lo indica, la búsqueda exhaustiva intenta evaluar cada una de todas las posibles soluciones en el espacio de búsqueda hasta que la mejor solución local es encontrada. Esto significa que si no sabemos qué tan buena es la mejor solución encontrada hasta el momento, no hay forma de saber si ya ha sido encontrada a menos que se haya considerado todo el espacio de búsqueda. Lamentablemente este enfoque no es de utilidad para problemas del mundo real para los cuales el tamaño del espacio de búsqueda generalmente es enorme. Sin embargo, algoritmos exhaustivos (denominados también enumerativos) son interesantes en algunos aspectos. Primero y principalmente, estos son muy simples dado que el único requerimiento es la generación sistemática de todas las posibles soluciones al problema en cuestión. Además, existen formas en que se puede reducir el trabajo a realizar, tal vez, haciendo backtracking [153]. Por ejemplo, algunos algoritmos clásicos de optimización (branch-and-bound o el algoritmo A^*) están basados en búsqueda exhaustiva.

- Algoritmos de aproximación: Son una herramienta útil cuando estamos tratando problemas de optimización. Ellos producen soluciones no óptimas, pero estas soluciones tienen la garantía de estar por debajo/arriba de un porcentaje fijo en relación al óptimo correspondiente [128]. Más formalmente, sea P un problema de optimización el cual tiene asociada una función de costo c positiva (se asume problema de minimización) y cuya solución factible óptima es f^* . Sea \mathcal{A} un algoritmo, el cual, dada una instancia I de P , retorna una solución factible $f_{\mathcal{A}(I)}$. Luego \mathcal{A} se denomina un algoritmo ϵ -aproximado si se cumple lo siguiente:

$$\frac{|c(f_{\mathcal{A}(I)}) - c(f^*)|}{c(f^*)} \leq \epsilon \quad (1.2)$$

La característica principal de estos algoritmos es que garantizan una solución factible con un valor cercano al óptimo acotado por un factor ϵ en relación al óptimo global. Sin embargo, no siempre es posible encontrar un algoritmo ϵ -aproximado para cualquier problema, particularmente dentro de la clase de problemas considerados en esta tesis. Los problemas pueden dividirse en tres grandes categorías desde esta perspectiva: totalmente, parcialmente y no aproximables, según los valores de ϵ para los cuales se satisface la condición 1.2.

Para llegar a tal conclusión, es necesario desentrañar estructuras relevantes del problema (como lo pueden hacer los algoritmos exactos) a fin de encontrar técnicas algorítmicas capaces de explotarlos. Dichas estructuras son en general más complejas en comparación con aquellas consideradas, por ejemplo, en una heurística greedy.

- Búsqueda local: Este es uno de los métodos más exitosos para atacar problemas de optimización combinatorios. La búsqueda local usa el concepto de estructura discreta para representar una solución y una función de *vecindario* para moverse a través del espacio de búsqueda [1]. La idea general de este enfoque es ir reemplazando

la solución corriente por una solución mejor, encontrada en el vecindario. Este proceso es repetido hasta que una solución localmente óptima es obtenida. La figura 1.3 muestra una posible trayectoria realizada por un algoritmo de búsqueda local sobre el espacio de búsqueda hasta llegar a un óptimo local.

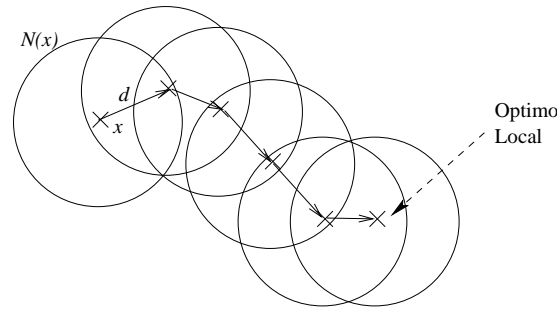


Figura 1.3: Trayectoria descrita por un algoritmo de búsqueda local acorde a una vecindad establecida. Es este caso se puede observar que la vecindad $N(x)$ incluye a todos los puntos que se encuentran a una distancia fija d de cada punto visitado x .

- Programación Lineal: Un problema en programación lineal requiere encontrar un extremo, por ejemplo el máximo, de una combinación lineal de variables,

$$f(\mathbf{x}) = \sum_{i=1}^n c_i x_i,$$

con $c_i \in \mathbb{Z}$ y sujeto a las restricciones primarias,

$$x_1 \geq 0, \dots, x_n \geq 0,$$

y también sujeto a $m = m_1 + m_2 + m_3$ restricciones adicionales; m_1 de la forma:

$$\sum_{j=1}^n a_{ij} x_j \leq b_j, \quad (b_j \leq 0), \quad i = 1, \dots, m_1,$$

m_2 de la forma:

$$\sum_{j=1}^n a_{ij} x_j \geq b_j, \quad (b_j \leq 0), \quad i = m_1 + 1, \dots, m_1 + m_2,$$

y m_3 de la forma:

$$\sum_{j=1}^n a_{ij}x_j = b_j, \quad (b_j \leq 0), \quad i = m_1 + m_2 + 1, \dots, m,$$

donde $a_{ij} \in \mathbb{Z}$ para $i = 1, \dots, m$ y $j = 1, \dots, n$. Cualquier vector \mathbf{x} que satisface todas las restricciones es denominada una solución factible y el objetivo es encontrar un vector factible que arroje el mejor resultado en término de la función de evaluación. El método simplex [67] es una técnica que procesa una solución completa. El mejor vector \mathbf{x} es construido en una serie de operaciones que toman ventaja de las características lineales del problema. Potencialmente, existen restricciones de igualdad y desigualdad impuestas a la función de evaluación, pero dado que la función de evaluación es lineal, el óptimo se encuentra en uno de los vértices del simplex (o sea la región factible), o en un caso degenerado, en algún lugar a lo largo del límite entre la región factible y no factible. La característica destacada del método simplex es que garantiza encontrar el mejor vértice del simplex.

- Algoritmos greedy: Este es un algoritmo que ataca el problema a través de la construcción de una solución completa en una serie de pasos, siendo su simplicidad una de las principales razones para usarlo. El procedimiento seguido por este enfoque es muy simple: asignar los valores a todas las variables de decisión una por una y en cada paso tomar la mejor decisión sin importar el efecto (no necesariamente positivo) sobre el valor objetivo de la solución completa. Este enfoque asume la existencia de una heurística para realizar dichas decisiones de manera tal que informe sobre el mejor posible movimiento en cada paso. Por esta razón es llamada greedy. Sin embargo, es claro que este enfoque es limitado en cuanto a la visión global del espacio de búsqueda dado que la toma de decisiones óptimas en cada paso en forma separada y localmente, no siempre

retorna la solución óptima global [2, 128]. También es importante señalar que este tipo de método es generalmente usado como base o componente de métodos heurísticos más avanzados (frecuentemente llamados metaheurísticas). Por otro lado, la manera de construir la solución paso a paso de este método está directamente vinculada a una de las metaheurísticas objeto de estudio de esta tesis. Dicha metaheurística es aquella basada en el comportamiento de las colonias de hormigas, la cual, como será explicado en el próximo capítulo, realiza de manera similar un proceso paso a paso para construir las soluciones. Sin embargo, existen diferencias sustanciales entre estos dos enfoques. Mientras que la metaheurística basada en colonia de hormigas es un proceso iterativo, poblacional y probabilístico, el enfoque greedy es determinístico, no iterativo y construye una única solución.

- **Dividir y Conquistar:** Esta técnica [4] consiste en dividir un problema P de gran complejidad en problemas más pequeños. Por lo tanto el problema original P es reemplazado por una colección de subproblemas, cada uno de los cuales a su vez es dividido en una serie de subproblemas y así siguiendo. Este proceso es generalmente realizado de una manera recursiva hasta que se alcanzan tipos de problemas lo suficientemente simples que pueden ser resueltos trivialmente. A fin de obtener la solución global del problema original P , el algoritmo comienza a combinar las distintas soluciones para construir la solución a un subproblema mayor y así siguiendo hasta llegar al problema original. Es importante destacar que para esta técnica, cuando es aplicable, es posible lograr una reducción importante en el tiempo de computación a través de una implementación paralela.
- **Programación Dinámica:** El término *dinámico* fue usado en los comienzos para identificarlo como un enfoque útil para problemas en

los cuales el tiempo juega un rol importante y en el cual el orden de las operaciones puede ser crucial [128]. Este enfoque trabaja de acuerdo al principio de encontrar una solución global operando sobre un punto intermedio situado entre un punto corriente y el punto a donde se quiere llegar. El procedimiento es recursivo. En dicho procedimiento, el próximo punto intermedio es elegido como una función de los puntos ya visitados. Un problema prototipo para el cual es aplicable programación dinámica tiene las siguientes propiedades:

- Puede ser descompuesto en una secuencia de decisiones realizadas en cada etapa.
- En cada etapa hay varios estados posibles.
- Una decisión nos lleva de un estado en una etapa a otro estado en otra etapa.
- La mejor secuencia de decisiones (conocida como *política*) en cualquier etapa es independiente de las decisiones realizadas en etapas anteriores.
- Existe un costo bien definido para ir desde un estado a otro estado a través de las distintas etapas. Inclusive existe una relación recursiva para elegir la mejor decisión a realizar.

Este método puede ser aplicado comenzando en el objetivo y trabajando hacia atrás desde el estado actual. Esto es, se determina la mejor decisión que puede ser realizada en la última etapa. Una vez determinada dicha decisión, se alcanza el estado previo correspondiente y así sucesivamente hasta que el estado inicial es alcanzado. En otras palabras, supongamos que se necesita realizar una secuencia de n decisiones para resolver un problema de optimización combinatoria, sean éstas D_1, D_2, \dots, D_n . Luego, si la secuencia completa es óptima, las últimas k decisiones, $D_{n-k+1}, D_{n-k+2}, \dots, D_n$, deben

ser óptimas. Esto significa que la secuencia final de una secuencia óptima, deber ser óptima, lo cual es referido como principio de optimalidad [128].

- **Branch-and-Bound:** Este método está basado en la idea de enumeración inteligente de todos los puntos factibles de un problema de optimización combinatorio. El término inteligente se refiere a la forma en que el algoritmo trabaja dado que en un particionado sucesivo del espacio de búsqueda, un importante número de soluciones no son consideradas. El término “branch” se refiere el proceso de particionado, mientras que “bound” se refiere a los límites inferiores⁹ (*lower bounds*) que son usados para construir una prueba de optimalidad para evitar realizar búsqueda exhaustiva. Si por ejemplo el espacio de búsqueda es visualizado como un árbol, la heurística involucrada en el enfoque *branch-and-bound* poda al árbol eliminando de éste aquellos subárboles (*branches*) que no son de interés [67]. Aunque este método ha mostrado ser eficaz en la resolución de muchos problemas de optimización, su principal limitación está dada por el crecimiento exponencial de la memoria requerida a medida que crece el tamaño de las instancias consideradas.
- **Algoritmo A*:** Esta técnica es una extensión de la búsqueda denominada *best-first*. En este caso el espacio de búsqueda es organizado como un árbol y las decisiones están orientadas a determinar la mejor (más eficiente) manera en cómo recorrerlo hasta encontrar la solución buscada. Es sabido que la eficiencia del algoritmo *best-first* está directamente relacionada con la calidad de la heurística usada para expandir los nodos durante la búsqueda. En consecuencia, una heurística bien diseñada debería demandar más atención. Luego, en el proceso de evaluar una solución parcialmente construida, digamos q , se deberían tomar en cuenta sus dos componentes: (1) el mérito

⁹Para un problema de minimización.

de las decisiones ya realizadas, $c(q)$, y (2) el potencial inherente de las decisiones remanentes, $h(q)$. Así, la función de evaluación *eval* para una solución parcial q está dada por $eval(q) = c(q) + h(q)$. En general, la formulación del problema provee una medición exacta de q . Sin embargo, la estimación de la calidad potencial de la decisión remanente (la heurística h) puede ser mucho más difícil de encontrar. Un criterio particular para definir la heurística h está basado en lo que se denomina *admisibilidad*¹⁰ la cual caracteriza al algoritmo A^* y que garantiza una solución globalmente óptima.

- Metaheurísticas: este tipo de métodos están siendo ampliamente usados en la actualidad para resolver problemas de optimización de gran interés práctico. Las metaheurísticas involucran una diversidad de técnicas y conceptos debido a lo cual no existe en general una definición precisa y compartida. La siguiente es una definición de metaheurísticas tomada de <http://www.metaheuristics.net>, una red que agrupa a investigadores relacionados al tema en cuestión:

Una metaheurística es un conjunto de conceptos que pueden ser usados para definir métodos heurísticos que pueden ser aplicados a una amplia variedad de problemas. En otras palabras, una metaheurística puede ser vista como un marco general algorítmico, el cual puede ser aplicado a diferentes problemas de optimización con mínimos cambios para ser adaptado a un problema específico.

Ésta puede ser considerada una definición circular dado que la misma involucra el término *métodos heurísticos*. Con el objeto intentar dar una definición más precisa del término *metaheurísticas*, es importante repasar algunas de las acepciones más conocidas del término *heurística*:

¹⁰Una heurística se denomina *admisible* si ésta nunca sobre-estima al costo de alcanzar el objetivo.

La palabra “heurística” se deriva del griego *heuriskein*, que significa “encontrar” o “descubrir”. El significado del término ha variado históricamente. Algunos han usado el término como un antónimo de “algorítmico”. Por ejemplo, Newell et al. [6] dicen:

A un proceso que puede resolver un cierto problema, pero que no ofrece ninguna garantía de lograrlo, se le denomina una “heurística” para ese problema.

Las heurísticas fueron un área predominante en los orígenes de la Inteligencia Artificial. Actualmente, el término suele usarse como un adjetivo, refiriéndose a cualquier técnica que mejore el desempeño en promedio de la solución de un problema, aunque no mejore necesariamente el desempeño en el peor caso [154].

La siguiente definición, proporcionada por Reeves [142], permite también englobar en forma precisa los métodos estudiados en esta tesis:

Una heurística es una técnica que busca soluciones buenas (es decir, casi óptimas) a un costo computacional razonable, aunque sin garantizar factibilidad u optimalidad de las mismas. En algunos casos, ni siquiera puede determinar qué tan cerca del óptimo se encuentra una solución factible en particular.

Sin embargo, el término metaheurísticas, de uso más reciente, tendría por objeto ampliar la definición anterior en cuanto a los procesos subyacentes involucrados en la “búsqueda de soluciones casi óptimas”. Por ejemplo, en “Handbook of Metaheuristics” (Glover et al. [73]) se presentan un veintena de métodos englobados bajo el nombre de metaheurísticas donde dicho término es definido de la siguiente manera y es el que hemos adoptado para referirnos a los métodos estudiados aquí:

Las metaheurísticas son métodos que integran de diversas maneras, procedimientos de mejora local y estrategias de alto nivel para crear un proceso capaz de escapar de óptimos locales y realizar una búsqueda robusta en el espacio de búsqueda. En su evolución, estos métodos han incorporado diferentes estrategias para evitar la convergencia a óptimos locales, especialmente en espacios de búsqueda complejos.

Aunque las metaheurísticas en general presentan un bajo nivel de desarrollo desde el punto de vista matemático, tales procesos son válidos en muchas situaciones prácticas para las cuales se han obtenido resultados muy interesantes.

Los enfoques metaheurísticos (principal tópico del próximo capítulo) incluyen muchas variaciones y muchas de ellas han sido propuestas basadas en procesos observados en la naturaleza y de esta manera relacionándose con otras disciplinas tales como física, estadística, evolución biológica, neurofisiología y comportamiento de algunos sistemas biológicos entre otras. Ejemplos de metaheurísticas incluyen Simulated Annealing (SA), Tabu Search (TS)¹¹, Búsqueda Local Iterada (sus siglas en Inglés ILS), Algoritmos Evolutivos (AEs), Optimización basada en Colonia de Hormigas (ACO: siglas en Inglés para Ant Colony Optimization), etc.

1.6. Evaluación de metaheurísticas

En general, para las metaheurísticas, no existen pruebas formales de cotas teóricas inferiores para el peor caso en relación al valor óptimo, como lo tienen por ejemplo los algoritmos de aproximación [128]. Por otro lado, si un análisis probabilístico asume una función de distribución de probabilidades sobre todas las posibles instancias, éste permitiría estudiar el caso-promedio (average-case). Sin embargo, es difícil en general,

¹¹Utilizaremos los términos en Inglés de aquí en adelante para *Simulated Annealing* y *Tabu Search*.

decir cómo debería ser una instancia promedio de un determinado problema, particularmente para el tipo de problemas que son considerados en esta tesis.

Un método simple para evaluar métodos metaheurísticos es a través de *estudios experimentales* de la misma forma que que son encontrados en la literatura relacionada con la aplicación de estos métodos a problemas de optimización.

Un estudio experimental se basa en una experiencia computacional en la cual un algoritmo es evaluado sobre un conjunto de instancias del problema. Instancias de tipos y tamaños diferentes son por lo general, generadas aleatoriamente de manera tal que la performance de una determinada metaheurística pueda ser observada bajo diferentes condiciones. Inclusive, para ciertos tipos de instancias y problemas es posible conocer *a priori* el valor óptimo o tal vez un valor esperado dado por cotas establecidas para la evaluación de la mejor solución. Esto es posible dado que los métodos usados para crearlas siguen determinadas reglas. Un ejemplo de esta situación se verifica cuando ciertas instancias son construidas siguiendo una estructura particular, lo que no sólo permite conocer *a priori* su solución óptima y su correspondiente valor objetivo, sino también poder incrementar su tamaño en un factor arbitrario con el consecuente cambio en el valor óptimo de la mejor solución de la instancia “escalada”.

Problemas estándar usados como referentes (benchmark), si están disponibles, son usados frecuentemente como instancias para ser cotejadas dado que sus valores óptimos (o mejor conocidos al momento) son generalmente de dominio público. El análisis empírico basado en benchmarks es muy común entre los desarrolladores de metaheurísticas, dada su utilidad cuando se reportan resultados de una nueva metaheurística o de una nueva versión de una existente a fin de comparar su desempeño contra el de técnicas ya conocidas. La ventaja del estudio experimental es que las comparaciones (ya sea contra el óptimo o el mejor valor conocido) son

precisas y seguras para una instancia de un problema dado. En suma, a través de la verificación de una metaheurística sobre un amplio rango de instancias de diferentes problemas, se espera que un cierto número de conclusiones pueda ser alcanzado en relación a cuan bien trabaja en general y bajo qué condiciones. También es importante remarcar que las comparaciones entre el comportamiento de varios métodos pueden realizarse en forma directa o indirecta, dependiendo de los objetivos principales del estudio experimental. Se dice que la comparación es realizada en forma directa, cuando se tiene acceso o bien se desarrolla el software de los métodos contra los cuales se pretende hacer la comparación a fin de replicar estudios experimentales previos o bien realizar nuevos estudios que incluyan, por ejemplo, instancias de distinto tipo y/o tamaño para un determinado problema y poder analizar, entre muchas otras cosas, los tiempos de ejecución sobre computadoras de idéntica capacidad de memoria y/o procesamiento. Sin embargo, la comparación directa puede ser problemática dado que no siempre es posible acceder al software o bien si la decisión es realizar una implementación nueva, la misma puede no ser la mejor lo que daría lugar a una comparación injusta. El enfoque indirecto, si bien es más limitado, nos permite sin embargo, llevar a cabo comparaciones interesantes, particularmente desde la perspectiva de la calidad de los resultados alcanzados y en muchos casos, la posibilidad de realizar comparaciones que son independientes de las implementaciones en particular y que están relacionada con el número de puntos del espacio de búsqueda visitados para alcanzar los resultados que cada método ha producido (esto último es posible cuando los reportes incluyen este tipo de conteos y está claramente explicado la manera en que se realizan).

Luego, las contribuciones de una metaheurística nueva o mejorada deben incluir lo siguiente [15]:

- Produce más rápidamente que otros enfoques soluciones de alta calidad.

- Identifica soluciones de más alta calidad que otros enfoques.
- Es menos sensible a las diferentes características de las instancias o ajuste de parámetros que otros enfoques.
- Es muy fácil de implementar.
- Puede ser aplicada directamente, o extendida para ser aplicada, a un amplio espectro de problemas académicos y del mundo real dada su robustez inherente.

Michalewicz en [114], establece que es posible seleccionar como medición de la performance la calidad de la mejor solución encontrada; el tiempo para alcanzarla; el mejor tiempo del algoritmo para alcanzar una solución aceptable; o la robustez del método. En muchos casos es esencial recordar el análisis de ciertos factores claves como la influencia del tamaño de las instancias en la calidad de las soluciones y también el esfuerzo computacional. Por último, pero no menos importante, un informe final debería contener toda la información necesaria para reproducir los resultados del estudio experimental.

Capítulo 2

Metaheurísticas aplicadas a problemas de optimización duros

2.1. Introducción

Las metaheurísticas están entre los desarrollos más recientes en métodos aproximados para resolver problemas de optimización complejos del mundo real que se encuentran en el área de comercio, ingeniería, industria y muchas otras áreas. Una metaheurística guía una heurística subordinada usando conceptos basados en inteligencia artificial, biología, matemática, ciencias físicas y naturales a fin de mejorar su efectividad. Este capítulo presenta una breve reseña de algunas de las más exitosas metaheurísticas desarrolladas en los últimos años. Estas son, entre otras, Simulated Annealing, Tabu Search y Algoritmos Evolutivos. Los dos primeros ejemplos son métodos de búsqueda a través de trayectorias (en este caso, versiones avanzadas de búsqueda local), mientras que el último representa un ejemplo de método basado en una población.

2.2. Metaheurísticas

Las metaheurísticas han evolucionado en forma dramática desde su invención en el comienzo de los 80s. Han sido exitosamente aplicadas a una gran variedad de problemas duros de índole práctico. Esta clase de

algoritmos pertenecen a la clase de algoritmos de aproximación que son diseñados para atacar problemas duros de optimización para los cuales las heurísticas tradicionales o métodos analíticos no son efectivos y eficientes. Las metaheurísticas proveen de una marco general que permite crear nuevos híbridos a través de la combinación de conceptos derivados de heurísticas clásicas, inteligencia artificial, evolución biológica, sistemas naturales, mecánica estadística, etc. Esta familia de enfoques incluyen, pero no están limitados a, algoritmos evolutivos, optimización basada en colonia de hormigas, búsqueda local, búsqueda local guiada, búsqueda local iterada, simulated annealing, tabu search, scatter search & path relinking, greedy randomized adaptive search process, redes neuronales, etc.

Es importante notar que existen pruebas de convergencia para algunas metaheurísticas bajo ciertas asunciones. Sin embargo, esas asunciones no pueden ser satisfechas o aproximadas en muchas aplicaciones prácticas. Por ende no es posible probar la (optimalidad) y en consecuencia sólo soluciones aproximadas pueden ser obtenidas. A pesar de esta desventaja, las metaheurísticas han demostrado ser muy útiles y eficaces en comparación con sus heurísticas subordinadas, para encontrar soluciones óptimas y/o cercanas a las óptimas correspondientes a una gran variedad de aplicaciones prácticas de optimización.

El resto de esta sección describe brevemente las siguientes metaheurísticas: simulated annealing, tabu search, algoritmos evolutivos y optimización basada en colonias de hormigas (ant colony optimization).

Es importante destacar que simulated annealing y tabu search son técnicas basadas en el procesamiento de una única solución para explorar el espacio de búsqueda, mientras que los algoritmos evolutivos y optimización basada en colonia de hormigas están basados en población. En el caso de algoritmos evolutivos, la población se refiere a un conjunto de individuos que representan, en general, la soluciones de problemas. Por otro lado, la población en un algoritmo basado en colonias de hor-

migas está representado por un conjunto de agentes que independiente y simultáneamente construyen las soluciones. En ambos casos, se realiza una exploración simultánea del espacio de búsqueda desde una perspectiva poblacional. En este sentido, simulated annealing y tabu search son sólo incluidas en la descripción con el objeto de dar una visión general de alguna de las metaheurísticas más actuales, sin embargo, el énfasis estará puesto en algoritmos evolutivos y los basados en el comportamiento de las colonias de hormigas, los cuales son objeto de estudio en la presente tesis.

2.3. Simulated Annealing

2.3.1. Introducción

Simulated Annealing — también conocida como templado de Monte Carlo, enfriamiento estadístico (statistical cooling), y algoritmo de intercambio estadístico — pertenecen a la clase de algoritmos de búsqueda local que son conocidos como algoritmos con umbrales. Estos algoritmos juegan un rol especial dentro del campo de búsqueda local por dos razones. Primero, han mostrado ser muy exitosos cuando ha sido aplicados a un amplio espectro de problemas prácticos. Segundo, algunos algoritmos con umbral, tal como simulated annealing, poseen una componente estocástica que facilita un análisis teórico de su convergencia asintótica—Cadenas de Markov, por ejemplo [1].

La clase general de algoritmos con umbral es presentado en la Figura 2.1. El procedimiento “Inicializar” selecciona del espacio de búsqueda una solución inicial. Luego el procedimiento “Genera” a su vez, selecciona una solución desde la vecindad ($\mathcal{N}(i)$) de la solución actual. La detención del algoritmo está controlada por “condicion_de_parada”. Puede observarse que este tipo de algoritmos continuamente considera soluciones vecinas de la solución actual y compara la diferencia en costo (acorde a f) entre dichas soluciones y un umbral (representado por t_k). Si la

diferencia entre los costos está por debajo del umbral, la solución del vecindario (j) reemplaza a la solución actual. Caso contrario, la búsqueda continua con la solución corriente inalterada. La secuencia t_k muestra una posible secuencia de umbrales a ser usados durante el proceso de búsqueda.

Existen tres tipos de algoritmos con umbral [1], dependiendo de la naturaleza del umbral t_k usado.

```

Algoritmo-Umbral;
{
     $i = \text{Inicializar}()$ 
     $k = 0$ 
    do
        Generar(  $j$  a partir de  $\mathcal{N}(i)$  );
        if (  $f(i) - f(j) < t_k$  )  $i = j$ ;
         $k = k + 1$ ;
    while ( !condicion_de_parada );
}

```

Figura 2.1: Algoritmo general basado en umbrales.

- *Mejora Iterativa*: $t_k = 0$, $k = 0, 1, 2, \dots$, es decir, un variante de la búsqueda local greedy clásica.
- *Umbral de aceptación*: $t_k = c_k$, $k = 0, 1, 2, \dots$; donde $c_k > 0$, $c_k \geq c_{k+1}$, y $\lim_{k \rightarrow \infty} c_k = 0$. Así, soluciones colindantes con costos mayores son aceptadas pero de una manera limitada. Durante la ejecución del algoritmo, los valores del umbral son disminuidos gradualmente hasta llegar a 0 en algunos casos.
- *Simulated annealing*: t_k = una variable aleatoria con un valor esperado $E(t_k) = c_k \in R^+$, $k = 0, 1, 2, \dots$. Cada t_k sigue una distribución de probabilidades F_{c_k} sobre R^+ . Este tipo particular de algoritmo

usa umbrales aleatorios con valores que varían entre 0 e ∞ . Luego la probabilidad de un umbral t_k de alcanzar un valor $y \in R^+$ está dado por $P_{c_k}\{t_k \leq y\} = F_{c_k}(y)$. Esto implica que cada solución colindante puede ser elegida con una determinada probabilidad para reemplazar a la solución corriente.

El origen de simulated annealing y la elección del criterio de aceptación de la nueva solución generada se basa en el proceso físico de enfriamiento de metal [98]. Esto es, en la materia condensada, el enfriamiento es un proceso termal a través del cual se alcanza estados de energía mínima de un sólido en un baño de calor, el cual consiste en los siguientes dos pasos:

1. La temperatura en el baño de calor es incrementada hasta un valor máximo al cual el sólido se funde;
2. luego la temperatura es disminuida cuidadosamente hasta que las partículas del sólido se re-organizan para conformar un reticulado altamente estructurado y la energía del sistema es mínima.

El proceso de templado puede ser modelado exitosamente en una computadora a través de métodos de simulación basado en técnicas de Monte Carlo. Metropolis y otros [111] propusieron un algoritmo muy simple para simular la evolución de un sólido en un baño de calor hasta alcanzar el equilibrio termal. Este algoritmo, basado en técnicas de Monte Carlo, genera una secuencia de estados del sólido de la siguiente manera:

Dado i , el estado corriente del sólido con energía E_i , el siguiente estado j es generado a través de la aplicación de un mecanismo de perturbación que transforma el estado corriente en el próximo estado aplicando una pequeña distorsión, como por ejemplo, desplazando una única partícula. La energía en el próximo estado es E_j . Si la diferencia $E_j - E_i$ es menor o igual a 0, el estado j es aceptado como el estado corriente. Si la energía es mayor que 0, el estado j es aceptado con una probabilidad dada por

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

donde T denota la temperatura del baño de calor y k_B es una constante física conocida como la constante de Boltzmann. La regla de aceptación dada previamente es conocida con el criterio de Metropolis, y el algoritmo correspondiente como algoritmo de Metropolis.

Si la temperatura es disminuida lentamente, el sólido puede alcanzar el equilibrio térmico para cada valor de temperatura. El equilibrio térmico está caracterizado por la distribución de Boltzmann, el cual relaciona la probabilidad de estar en el estado i con energía E_i y temperatura T , y es dado por

$$P_T\{X = i\} = \frac{\exp(-E_i/k_B T)}{\sum_j \exp(-E_j/k_B T)}$$

donde X es una variable aleatoria que denota el estado corriente del sólido y la sumatoria se extiende sobre todos los estados.

Aunque la distribución de Boltzmann juega un rol esencial en el análisis de simulated annealing, el criterio de Metropolis es aplicado para generar una secuencia de soluciones de un problema particular de optimización combinatoria. Para este propósito, se asume una analogía entre un sistema físico consistente de muchas partículas y un problema de optimización con las siguientes equivalencias:

- Las soluciones de un problema de optimización combinatoria son equivalentes a los estados de un sistema físico.
- El costo de la solución es equivalente a la energía del estado.

Una característica distintiva de simulated annealing es que a pesar de aceptar nuevas soluciones basado en mejoras en el costo, también acepta, de una manera limitada, soluciones con un incremento en el costo. Inicialmente, con valores grandes para T , soluciones con un costo

incremental son frecuentemente aceptadas, a medida que el valor de T disminuye, raramente tal tipo de soluciones son aceptadas, y cuando T se acerca a cero, sólo aquellas soluciones que mejoran la anterior son aceptadas. En general, el algoritmo para simulated annealing (SA) es mostrado en la figura 2.2.

```

procedure SA
begin
   $t = 0$ 
  generar una solución en forma aleatoria (estado  $E_i$ )
  evaluar  $E_i$ 
  repeat
    repeat
      Identificar la vecindad  $\mathcal{N}(E_i)$ 
      Seleccionar  $E_j \in \mathcal{N}(E_i)$ 
      if  $eval(E_i) > eval(E_j)$ 
        then  $E_i = E_j$  /*  $E_j$  es ahora el nuevo estado */
      else if  $random[0, 1) < e^{\frac{E_j - E_i}{k_B \cdot T}}$  then  $E_i = E_j$ 
    until (condición-de-terminación)
     $T = g(T, t)$ 
     $t = t + 1$ 
  until (criterio-de-parada)
  output  $E_i$  /* la mejor solución encontrada */
end

```

Figura 2.2: Una implementación general para SA

2.3.2. Simulated Annealing, ¿cómo trabaja?

El algoritmo de simulated annealing puede ser modelado matemáticamente usando la teoría de cadenas finitas de Markov [1, 143]. La convergencia asintótica de simulated annealing puede ser probada basándose en un modelo en el cual el algoritmo es visto como una secuencia de cadenas de Markov de longitud infinita. El modelo puede ser homogéneo o

heterogéneo. En el caso homogéneo, el valor del parámetro de control es independiente de k ; esto es, $c_k = c$ para todo k . Para el caso heterogéneo, existe una secuencia diferente de c_k para $k = 0, 1, \dots$ la cual satisface las siguientes condiciones,

$$\lim_{k \rightarrow \infty} c_k = 0$$

, y

$$c_k \geq c_{k+1}, \quad k = 0, 1, \dots$$

El algoritmo SA, bajo condiciones no muy severas, converge en probabilidad a un conjunto de soluciones óptimas. Sin embargo, la convergencia asintótica hacia dicho conjunto solamente es alcanzada después de un número infinito de transiciones [143] lo cual lo torna impráctico cuando es aplicado a un problema para el cual se requiere de una solución en un tiempo razonable. Por lo tanto, en cualquier implementación real para la cual se espera una respuesta en tiempo razonable, se recurre a aproximaciones de la convergencia asintótica. Varios autores han investigado diferentes posibilidades para acelerar la convergencia de simulated annealing para problemas específicos teniendo en cuenta la estructura combinatoria del problema en cuestión [1, 143].

Lamentablemente, implementaciones limitadas a un tiempo finito no dan la garantía de encontrar una solución óptima, pero puede resultar un algoritmo que sea mucho más rápido sin comprometer significativamente la calidad de la solución encontrada. En este contexto, un algoritmo SA es obtenido a través de la generación de una secuencia de cadenas Markov de longitud finita usando valores decrecientes del parámetro de control. Por esta razón, el conjunto de parámetros debe ser convenientemente especificados para influir positivamente en la convergencia del algoritmo. Dichos parámetros son combinados en lo que es denominado la planificación del enfriamiento (schedule cooling).

Definición 2.1 *Una planificación de enfriamiento es una secuencia finita de valores de parámetros de control, un número finito de transiciones para cada valor del parámetro de control. Más precisamente, es especificado por*

- *un valor inicial del parámetro de control c_0 ,*
- *una función de decremento para disminuir el valor del parámetro de control,*
- *un valor final para el parámetro de control especificado por un criterio de parada*
- *una longitud finita de cada cadena de Markov*

Al comenzar el proceso la temperatura es tan alta que a lo sumo todos los intercambios (nuevos estados) son aceptados, lo que produce simplemente un camino aleatorio. Esta parte de la búsqueda es más lenta dado que un mayor número de aceptación significa que un gran número de modificaciones de la solución deben ser ejecutados. Por el contrario, cuando la temperatura tiende a bajar, sólo pequeños movimientos son posibles y menos incrementos en los costos son aceptados, de esta manera el algoritmo degenera en una versión muy lenta de un algoritmo descendente. Consecuentemente podría ser ventajoso explorar solamente cuando la temperatura se encuentra entre los valores extremos del rango considerado y usando el tiempo ahorrado para permitir un enfriamiento más lento. En relación a éste y otras cuestiones de índole práctica, la búsqueda de planificaciones de enfriamiento adecuadas ha sido un tópico de suma importancia en diferentes líneas de investigación. Por ejemplo, en [1] han sido investigados esquemas de enfriamiento de tiempo finito óptimos, donde óptimos se refiere a mejor costo promedio obtenido en ese tiempo finito. La mayoría de los trabajos existentes en esta área está relacionado con esquemas de enfriamiento basados en alguna heurística particular. Aarts y otros, en [1] distinguen tales planificaciones en dos

clases: planificaciones estáticas y dinámicas. En el esquema estático, los parámetros se mantienen con valores fijos a través de la ejecución del algoritmo. En el dinámico, los valores de los parámetros son cambiados en forma adaptativa durante la ejecución del algoritmo.

2.3.3. Aplicaciones de SA

Al presente, una inmensa cantidad de artículos han sido publicados en los cuales se reportan aplicaciones muy variadas de SA. Muchos de esos estudios han llevado a realizar modificaciones del algoritmo tales como la incorporación de funciones de penalidad, probabilidades alternativas para la generación y aceptación de soluciones, aspectos específicos de la implementación, versiones paralelas y muchas otras.

2.4. Tabú Search

2.4.1. Introducción

Tabú Search (TS) fue introducida por Glover [72]. TS es un procedimiento heurístico para encontrar buenas soluciones a problemas de optimización combinatoria. Los antecedentes de TS están vinculados a métodos diseñados para cruzar los bordes del espacio factible o de optimalidad local normalmente tratado como barreras los que sistemáticamente imponen una flexibilización en las restricciones de manera tal de permitir la exploración de áreas que de otra manera estarían prohibidas. Según la definición del diccionario (ver referencia a diccionario en español) ‘tabú’ significa “poseer un peligroso poder sobrenatural, de uso o contacto prohibido”. Ciertamente, TS no está involucrado con poderes sobrenaturales o inclusive con cuestiones morales, en cambio está vinculado al hecho de imponer algunas restricciones que operan de manera diferente con el objetivo de dirigir la búsqueda a nuevas áreas del espacio de búsqueda. Básicamente, la filosofía de TS está relacionada a la deri-

vacación y explotación de una colección de principios para la resolución de problemas de una manera inteligente. Un elemento fundamental subyacente a TS es el uso de una memoria flexible que representa el proceso de crear y explotar ciertas estructuras para tomar ventaja de la historia. Dicha estructura de memoria opera en cuatro dimensiones diferentes; *uso-reciente*¹, *frecuencia*, *calidad* e *influencia*. El siguiente ejemplo nos permitirá entender el comportamiento de TS:

2.4.2. TS aplicada al Problema del Viajante de Comercio

El Problema del Viajante de Comercio o Traveling Salesman Problem (TSP) como muchos otros problemas que aceptan una permutación como posible solución, forma un clase muy importante de problemas de optimización combinatoria. Además, los problemas de optimización ofrecen un vehículo útil para demostrar algunas consideraciones a ser enfrentadas en el dominio de los problemas combinatorios.

TS opera bajo la premisa de que una *vecindad* puede ser construida para identificar soluciones adyacentes que pueden ser alcanzadas a partir de la solución corriente — es decir, debería ser posible definir operadores acorde a la representación de la solución con el propósito de realizar movimientos en el espacio de búsqueda. Un intercambio de a dos (*swap* entre ciudades) se usa frecuentemente para definir vecindades en problemas de permutación como TSP. Por ejemplo, la permutación (2, 5, 7, 3, 4, 6, 1) que es una solución para una instancia de TSP de 7 ciudades, tiene una vecindad de 21 elementos.

Un mecanismo muy importante para explotar la memoria en TS es clasificar un subconjunto de movimientos en una vecindad como prohibidos o tabú. Esta clasificación depende de la historia de la búsqueda los cuales pueden estar dados por el uso-reciente o frecuencia que ciertos

¹Este término corresponde al original en Inglés *recency*, el cual hace referencia a cuán recientemente ha sido usado un determinado movimiento o componente de una solución para la obtención de una nueva.

movimientos o componentes de la solución han participado en generar soluciones pasadas. Esto significa que en cualquier etapa, hay una solución actual que está siendo procesada y por ende existe una vecindad para la misma, sin embargo en dicha vecindad podrían existir soluciones tabú que no serán consideradas para una posible exploración.

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

Figura 2.3: Estructura de datos tabú que representa los movimientos tabú para TS.

Lamentablemente, una política de estas características puede ser muy restrictiva. Podría ocurrir que uno de los miembros de la vecindad tabú diera un muy buen puntaje de evaluación. Es decir que dicho puntaje podría ser mejor que todas las otras soluciones previas. Una posibilidad para tratar esto es hacer la búsqueda más flexible. Por ejemplo, considerando todas las soluciones de la vecindad, evaluarlas a todas y bajo circunstancias “normales” seleccionar una solución no tabú como la próxima solución. Pero en circunstancias que no son “normales”, es decir que si una solución tabú muy buena es encontrada en la vecindad, tal solución superior es tomada como la próxima solución actual. Esta excepción de la clasificación tabú ocurre cuando un criterio particular, llamado *criterio de aspiración*, es satisfecho. Para nuestro ejemplo, las cuatro iteraciones que siguen muestran el procedimiento tabú básico que emplea la restricción tabú de *módulo apareado* y el criterio de aspiración de la *mejor solución*.

Iteración 0

Punto corriente
costo del tour = 20

2	5	7	3	4	6	1
---	---	---	---	---	---	---

5 mejores movimientos

Swap	Valor
5,4	-6
7,4	-4
3,6	-2
2,3	0
4,1	-1

*

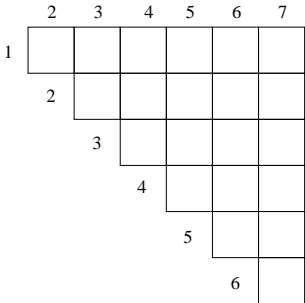


Figura 2.4: Estructura Tabu - Iteración 0

Supongamos que el tour inicial tiene un costo de 20, y la estructura de datos o memoria inicialmente está vacía. La tabla de la derecha, arriba, correspondiente a la iteración 0 muestra los movimientos swap candidatos (los mejores 5). La columna denominada Valor representa la disminución en el costo del tour si el movimiento swap es realizado entre el par de ciudades correspondientes. Para encontrar el tour mínimo local, se intercambian las ciudades 5 y 4.

Iteración 1

Punto actual
costo del tour = 14

2	4	7	3	5	6	1
---	---	---	---	---	---	---

5 mejores movimientos

3,1	-2
2,3	-1
3,6	1
7,1	2
6,1	4

*

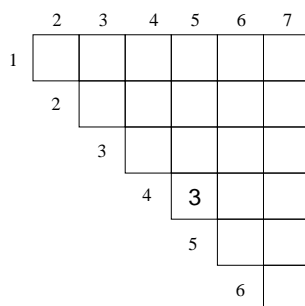


Figura 2.5: Estructura Tabú - Iteración 1

El nuevo tour tiene un costo de 14— es decir, el costo previo menos el valor de movimiento seleccionado. La estructura tabú muestra ahora que el intercambio (swap) de las posiciones de los módulos 4 y 5 estará prohibido por $t = 3$ iteraciones. El valor t representa la tenencia tabú, la cual resulta en una prohibición de 3 de 21 movimientos posibles. Ahora, en la iteración 1, el movimiento más beneficioso es intercambiar las ciudades 3 y 1, lo que hace disminuir el costo del tour en 2 unidades.

Iteración 2

Punto actual 5 mejores movimientos
costo del tour = 12

2	4	7	1	5	6	3
---	---	---	---	---	---	---

1,3	2	T
2,4	4	*
7,6	6	
4,5	7	T
5,3	9	

La nueva solución es la mejor solución encontrada hasta el momento con un costo de 12. Sin embargo ahora, dos de los posibles movimientos están clasificados tabú según lo indican las entradas con valores distintos de cero en la estructura tabú. Puede observarse (Iteración 2) que la entrada (4, 5) ha sido decrementada de 3 a 2, indicando que permanecerá como tabú por dos iteraciones más. Por otro lado, ninguno de los movimientos tiene un valor negativo asociado. Además, el movimiento que no produce mejoras que parecería ser más atractivo de realizar es

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

Figura 2.6: Estructura tabú - Iteración 2

el inverso de un movimiento realizado en una iteración previa. Por ende, es clasificado como tabú y el próximo movimiento lo que hace es intercambiar las ciudades 2 y 4.

Iteración 3

Punto actual 5 mejores movimientos
costo del tour = 14

2	5	7	3	4	6	1
---	---	---	---	---	---	---

4,5	-6	T*
5,3	-2	
7,1	0	
1,3	3	T
2,6	6	

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

Figura 2.7: Estructura Tabú - Iteración 3

Después del último movimiento, el nuevo tour alcanza un costo de (14) debido a la existencia de un movimiento con valor positivo. la estructura tabu tiene ahora 3 movimientos clasificados como tabú, con diferentes grados respecto a dicha condición. La lista de movimientos

candidatos tiene al movimiento (4, 5) como el mejor pero aún está considerado como tabú según lo indica la estructura respectiva. Sin embargo, si este movimiento es realizado, se encuentra un tour que es de menor costo que cualquiera de los encontrados anteriormente. En consecuencia, se usa el criterio de aspiración para invalidar la clasificación tabú de este movimiento y se le elige como el mejor de la iteración actual.

Iteración 4

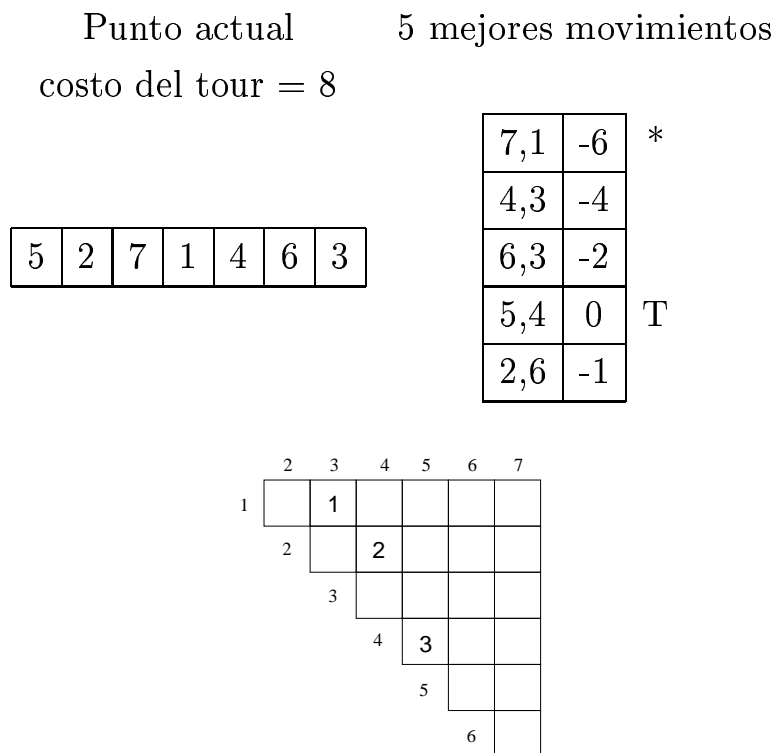


Figura 2.8: Estructura Tabú - Iteración 4

En la iteración 4, el algoritmo obtiene un tour con costo 8, la mejor solución hasta el momento.

En algunas situaciones, puede ser deseable incrementar el porcentaje de movimientos disponibles que reciben una clasificación tabú. Esto podría ser alcanzado ya sea, a través del incremento del grado de la condición tabú o cambiando las restricciones, es decir, prevenir que algunos movimientos puedan ser realizados.

En la figura 2.9 se muestra una descripción general de una implementación particular del algoritmo TS para TSP. La sección de código

“Seleccionar el mejor movimiento admisible de $\mathcal{N}(\mathcal{T})$ ” toma en cuenta la lista tabú para la selección de un movimiento particular.

```

procedure TS
begin
    tries = 0
    repeat
        generar un tour  $T$  (La solución inicial)
        cont = 0
        repeat
            Identificar la vecindad  $\mathcal{N}(T)$ 
            Seleccionar el mejor movimiento admisible de  $\mathcal{N}(T)$ 
            Realizar el movimiento
            Modificar la lista tabú y otras variables
            if el nuevo tour es el mejor-hasta-el-momento
                para uno de los ‘intentos’
            then modificar la información del mejor tour local
            coun = cont + 1
        until = ITER
        intentos = intentos + 1
        if el tour actual es el mejor-hasta-el-momento
            (para todos los ‘intentos’)
        then modificar la información del mejor tour global
    until intentos = MAX_INTENTOS
end

```

Figura 2.9: Una implementación particular de TS para TSP.

2.4.3. Extensiones de Tabu Search

A diferencia de otras metaheurísticas, la aleatorización en TS es empleada solamente de una manera altamente restringida, con la premisa de que la búsqueda inteligente debería estar basada principalmente en una manera sistemática de realizar la guía a través del espacio de búsqueda. Es importante destacar, en relación a este punto, que muchas imple-

mentaciones de TS son totalmente determinísticas. Sin embargo, existen muchas variaciones de tabú search que incluyen: tabú search probabilística, tabú search reactiva [16], etc. Además, una cantidad de posibilidades han sido consideradas en relación a las restricciones tabú, tanto como el criterio de aspiración como así también el rol de la intensificación y diversificación [143]. La estrategia de intensificación se encarga de crear soluciones en forma agresiva promoviendo la incorporación de buenos atributos (explotación). A corto plazo, esto consiste en la incorporación de atributos que reciben la más alta evaluación según diferentes enfoques y criterios, mientras que en el mediano y corto plazo, esto consiste en la incorporación de atributos de soluciones de subconjuntos seleccionados de elite — esto significa que la búsqueda se concentra o está dirigida a subregiones definidas según los subconjuntos seleccionados. Por otro lado, la estrategia de diversificación, lo que no significa aleatorización, trata de generar soluciones que involucren composiciones de atributos que sean significativamente diferentes de aquellos encontrados previamente durante el proceso de búsqueda.

2.4.4. Aplicaciones de Tabú Search

Los problemas de scheduling proveen una de las áreas más fructíferas en relación a las técnicas heurísticas en general y a TS en particular. Así, para el Flow y Job Shop Scheduling y variantes de estos problemas existe un importante número de trabajos en donde TS fue aplicado con diferentes grados de éxito [143]. Ruteo de Vehículos es otro problema que ha recibido mucha atención de la comunidad de TS [124, 38, 167] como también el problema de Mochilas Múltiples [74], Asignación Cuadrática [63]², Optimización Multiobjetivo [85, 84], Satisfacción de restricciones [126, 135], entre otros. Además, una importante cantidad de versiones paralelas de este método han sido implementadas a fin de mostrar la importancia de la computación paralela en la resolución de problemas de

²Usado aquí en combinación con el enfoque ACO.

optimización combinatoria de gran tamaño y la flexibilidad del método para ser aplicada a distintos tipos de problemas.

2.5. Algoritmos Evolutivos

2.5.1. Introducción

La evolución puede ser vista como el resultado de la integración de cuatro procesos esenciales: reproducción, competencia, mutación y selección. La reproducción es llevada a cabo a través de la transferencia de un programa genético del individuo a su progenie. La mutación, en un sistema positivamente entrópico el cual garantiza que la replicación de errores ocurrirá necesariamente durante el proceso de transferencia. La competencia es una consecuencia de la expansión de las poblaciones en un espacio de recursos finitos. Y finalmente la selección, es el resultado inevitable a medida que las especies comienzan a sobrepasar la capacidad del espacio disponible. La evolución en sí misma puede ser vista como un proceso de optimización donde “optimización” no necesariamente implica “perfección”, sino más bien, soluciones funcionales altamente precisas para un problema en particular. Por lo tanto es muy natural intentar describir la evolución en términos de un algoritmo que pueda ser usado para resolver distintos tipos de problemas. Los algoritmos basados en la evolución natural son denominados *algoritmos evolutivos* (AEs) los que serán descritos en detalle en esta sección.

A continuación se introduce una breve reseña histórica acerca de los AEs. Alrededor de 1955, las computadoras digitales permitieron desarrollar programas para entender más adecuadamente el proceso de evolución [9, 10]. Box y sus colegas (1957, 1960, 1966, 1969) trabajaron sobre una técnica llamada *operación evolutiva* (EVOP) que fue aplicada en una planta de manufactura como un proceso de administración implementado a través de los votos de un comité de administradores técnicos. A través de EVOP, la planta de manufactura es vista como una especie en

evolución. Sin embargo, EVOP nunca fue implementado en una computadora como un proceso de simulación. Friedberg (1958) [64] y Friedberg et al. (1959) [65] describen un proceso evolutivo para programación automática. Al mismo tiempo, Bremermann [28, 29] presenta un algoritmo basado en evolución simulada para resolver problemas de optimización numérica y también incursionó en algunos desarrollos incipientes de la teoría de algoritmo evolutivos. Durante este periodo otros estudios fueron llevados a cabo, pero ellos se encontraron con un gran escepticismo. Finalmente, a mediados de los '60, tres formas principales de AEs fueron establecidos según se muestra en la Figura 2.10.

- *Estrategias Evolutivas* (EEs): un desarrollo conjunto llevado a cabo por a Bienert, Rechenberg y Schwefel, en Berlín (Rechenberg, 1965).
- *Programación Evolutiva* (PE): desarrollada por Lawrence Fogel en San Diego, California (Fogel et al., 1966)
- *Algoritmos Genéticos* (AGs): desarrollados en la Universidad de Michigan por John H. Holland (1967)

Cada una de estas subáreas se desarrollaron en forma independiente y en consecuencia, con historias paralelas bien diferenciadas. Además de estos tres métodos directrices, es importante destacar una variante de los AGs introducida por Koza [102, 101] en 1992 llamada *Programación Genética* (PG) la cual involucra una búsqueda guiada por la evolución en el espacio de todos los posibles programas. Sin embargo, en 1990 se organizó un foro de discusión (*Parallel Problem Solving from Nature*) con el objeto de unir los esfuerzos y promover la discusión entre los miembros de la comunidad de Computación Evolutiva y de otras metaheurísticas. Dentro de la taxonomía de la figura 2.10 aparece *Tierra* como una subárea de AGs. Esta es una propuesta de Thomas S. Ray <http://www.his.atr.jp/~ray/index.html>. Si bien existen una serie de publicaciones relacionadas, esta subárea no ha tenido un impacto importante en la comunidad de Computación Evolutiva desde su creación.

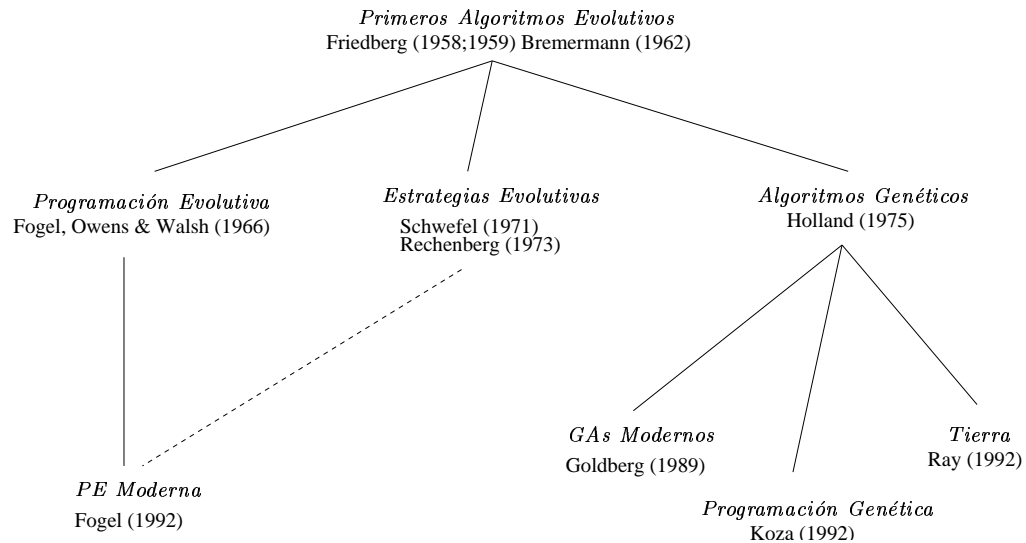


Figura 2.10: Taxonomía histórica común de la computación evolutiva.

2.5.2. Descripción General

La Programación Evolutiva, las Estrategias Evolutivas y los Algoritmos Genéticos comparten algunas características comunes:

- Utilizan un proceso de aprendizaje colectivo de una población de individuos, es decir, son métodos basados en población.
- Modelan la mutación y recombinación a través de un proceso aleatorio.
- Evalúan a los individuos en su ambiente a fin de obtener una medida de su calidad (valor de fitness del individuo). De esta manera, se permite al proceso de selección favorecer a los mejores individuos.

A pesar de dichas similitudes, cada instancia de un AE usa dichas características en forma diferente. Sin embargo, es posible dar un marco general a fin de describir un *algoritmo evolutivo básico* desde una perspectiva general.

Sea I un espacio arbitrario de individuos, $a \in I$ y $F : I \rightarrow R$ una función real que denota el *fitness* de los individuos. Esta función mide la calidad de la solución con respecto al problema bajo considera-

ción. Sean μ y λ los tamaños de la población de padres e hijos respectivamente. Luego, la población en la generación t viene dada por $P(t) = (a_1(t), a_2(t), \dots, a_\mu(t)) \in I^\mu$. La selección, mutación y recombinación (crossover) son considerados operadores $s : I^\lambda \rightarrow I^\mu$, $m : I^\kappa \rightarrow I^\lambda$, and $r : I^\mu \rightarrow I^\kappa$ que permiten transformar las poblaciones. Estos operadores dependen en general de parámetros adicionales para su aplicación ϕ_s , ϕ_m , y ϕ_r los cuales son característicos del operador y de la representación del individuo. Además, existe un proceso de inicialización que genera una población de individuos³. Una rutina de evaluación determina el valor de *fitness* de cada individuo de una población. Por último, un criterio de terminación es aplicado para determinar si el algoritmo debería detenerse o continuar. El algoritmo 2.11 representa un algoritmo evolutivo básico según las consideraciones previas.

Entrada: $\mu, \lambda, \Theta_\iota, \Theta_r, \Theta_m, \Theta_s$
 Salida:
 a^* , el mejor individuo durante la corrida, o
 P^* , la mejor población encontrada.

```

1   $t \leftarrow 0$ ;
2   $P(t) \leftarrow \text{inicializar}(\mu)$ ;
3   $F(t) \leftarrow \text{evaluate}(P(t), \mu)$ ;
4  while (  $\text{term}(P(t), \Theta_\iota) \neq \text{true}$  ) do
5       $P'(t) \leftarrow \text{recombinar}(P(t), \Theta_r)$ ;
6       $P''(t) \leftarrow \text{mutar}(P'(t), \Theta_m)$ ;
7       $F(t) \leftarrow \text{evaluar}(P''(t), \lambda)$ ;
8       $P(t+1) \leftarrow \text{seleccionar}(P''(t), F(t), \mu, \Theta_s)$ ;
9       $t \leftarrow t + 1$ ;
10 od
```

Figura 2.11: Descripción general de un Algoritmo Evolutivo.

Después de la inicialización de t (línea 1) y de la población $P(t)$ con un tamaño μ (línea 2), como así también la evaluación del *fitness* (línea

³Generalmente en forma aleatoria, pero existen otras alternativas.

3), se ingresa a la iteración. El criterio de terminación l podría depender de una variedad de parámetros, los cuales son sintetizados en el algoritmo a través de Θ_l . De manera similar, la recombinación (línea 5). mutación (línea 6), y la selección (línea 8) dependen de un número de parámetros adicionales del algoritmo. Mientras que $P(t)$ consiste de μ individuos, se asume que $P'(t)$ y $P''(t)$ tienen una cantidad de individuos κ y λ respectivamente. Cuando $\lambda = \kappa = \mu$, es el caso particular de los AGs con reemplazo generacional (ver sección 2.5.7 para más detalles). Cuando $\kappa = \mu$ es un caso particular de PE, sin recombinación, pero esto depende de la aplicación. También puede ocurrir que ya sea la mutación o recombinación estén ausentes en el bucle principal, es decir $\kappa = \mu$ sin recombinación y $\kappa = \lambda$ sin mutación. El operador de selección escoge μ individuos de $P''(t)$ de acuerdo a los valores de *fitness* $F(t)$, luego t es incrementado y nuevamente se ejecuta el cuerpo de la iteración. Los parámetros de entrada de este algoritmo evolutivo incluye los tamaños de población μ y λ así como los parámetros Θ_l , Θ_r , Θ_m , y Θ_s . Es importante notar que la recombinación puede estar dada según un mapeo de identidad, esto es, $P''(t) = P'(t)$ para aquellos AEs que no involucran este operador.

A continuación se explican los algoritmos evolutivos más representativos, vistos como instancias particulares del esquema general descrito anteriormente.

2.5.3. Estrategias Evolutivas

Este enfoque fue desarrollado en forma independiente por Rechenberg [140](1965) y Schwefel [160] (1965). Cada uno de ellos aplicaron el método a problemas de optimización de funciones continuas. A continuación damos un enfoque general para aplicar este método.

- El problema es definido en términos de la búsqueda de un vector x de valores reales de dimensión n asociado a una función $f(\mathbf{x}) : \mathbb{R}^n \rightarrow \mathbb{R}$.

- Se crea una población inicial de vectores padres \mathbf{x}_i , $i = 1, \dots, \mu$ en forma aleatoria considerando valores factibles en cada dimensión.
- Un vector hijo \mathbf{x}'_i es creado a partir de cada padre \mathbf{x}_i , $i = 1, \dots, \mu$, a través de un valor añadido en cada una de sus componentes según una variable aleatoria con distribución Gaussiana con $\mu = 0^4$ y una desviación estándar pre-seleccionada.
- La selección determina posteriormente cuales de los vectores mantener según un ranking de errores $f(\mathbf{x}_i)$ y $f(\mathbf{x}'_i)$, $i = 1, \dots, \mu$. Los vectores en P que poseen el menor error conformarán el conjunto de nuevos padres para la próxima generación.
- El proceso de generar nuevos vectores y seleccionar aquellos con el mejor valor objetivo, continúa hasta que una solución satisfactoria es encontrada o el tiempo de computación asignado ha expirado (esto se podría considerar como el máximo número de generaciones).

Las primeras versiones de las ESs estaban basadas en una población consistente de un individuo y un operador genético, la mutación. El individuo se representaba a través de un par de vectores reales de la siguiente manera:

$$v = (\mathbf{x}, \sigma)$$

donde el vector \mathbf{x} representa el punto en el espacio de búsqueda y el vector σ la desviación estándar. Un nuevo vector es obtenido como resultado de aplicar la mutación al vector \mathbf{x} según una distribución Gaussiana con $\mu = 0$ y una desviación estándar σ . Luego un nuevo individuo es obtenido según la fórmula:

$$\mathbf{x}^{t+1} = \mathbf{x}^t + N(0, \sigma)$$

⁴Aquí se hace referencia a la media de la distribución. A menos que se especifique lo contrario o se deduzca del contexto, mantendremos el significado anterior dado que es la nomenclatura usada en AEs en general y en EEs, en particular.

donde $N(0, \sigma)$ es un vector de números reales generados en forma independiente siguiendo una distribución Gaussiana, donde la variable t indica la generación corriente. Este enfoque fue denominado $(1 + 1)$ -ES dado que un único hijo es creado a partir de un único padre y ambos son puestos a “competir”, de manera tal que la solución de menor calidad es eliminada. Schwefel [161] (1981) incorporó el uso de múltiples padres e hijos en las ESs. Rechenberg [141] (1973) uso múltiples padres pero un único hijo era generado en cada generación. Otros enfoques más recientes han sido considerados, $(\mu + \lambda)$ y (μ, λ) . En la sección 2.5.7 nos referiremos en detalle a cada uno de estos métodos.

2.5.4. Programación Evolutiva

La Programación Evolutiva (PE) es un enfoque creado por Lawrence J. Fogel [61, 59, 60, 62, 62]. PE es una estrategia estocástica de optimización en donde el algoritmo optimiza comportamiento, parámetros y otras construcciones en forma iterativa.

La creación de PE fue motivada por la necesidad de generar un enfoque alternativo en inteligencia artificial. En este sentido, Fogel concibió el uso de la simulación de la evolución como una herramienta para desarrollar algunos aspectos de la inteligencia artificial sin considerar heurísticas, sino mas bien individuos creados a través de varias generaciones con un incremento en su inteligencia (evolución). Esta visión está basada en la observación que un comportamiento inteligente requiere la habilidad de un organismo para realizar predicciones correctas para un objetivo dado. El uso inicial de PE fue para evolucionar una población de máquinas de estado finito (autómatas finitos) las cuales eran expuestas a un entorno representado por una secuencia de símbolos tomados de un alfabeto de cardinalidad finita. En estos experimentos el problema fue definido de la siguiente manera: evolucionar un algoritmo que pueda operar sobre una secuencia de símbolos previamente observados. El objetivo era que la máquina de estado finito pudiera producir una símbolo de salida que

maximice el beneficio del algoritmo considerando el próximo símbolo que pudiera aparecer en el ambiente de acuerdo a una función de refuerzo bien definida [61].

La forma básica para el método PE involucra 3 pasos (El bucle repeat-until es ejecutado hasta que un umbral en el número de iteraciones es alcanzado o una solución adecuada es obtenida):

1. Generar una población inicial de soluciones en forma aleatoria. En PE el número μ de soluciones en una población es relevante en relación a la velocidad de optimización, sin embargo no existen respuestas definitivas en relación a cuántas soluciones son apropiadas y cuántas excesivas.
2. Cada solución es replicada en una nueva población. Cada uno de esas μ soluciones (hijos) son mutadas acorde a una distribución de tipos de mutación, que van desde muy extremas a otras que producen pequeños cambios. La severidad de la mutación puede ser juzgada sobre la base de un cambio funcional impuesto sobre los padres (aquí estamos hablando de un cambio en el comportamiento inducido por un cambio en la estructura del individuo, que pueden no tener relación, es decir, un pequeño cambio en la estructura puede llevar a un gran cambio en el comportamiento)
3. La calidad de cada solución generada es estimada a través del cómputo de su *fitness*. En general, un torneo estadístico es realizado para determinar las soluciones que serán retenidas para la próxima generación, aunque ocasionalmente es determinado en forma determinística. Tampoco existe el requerimiento de que el tamaño de la población deba permanecer constante, ni que sólo un único hijo sea generado a partir de un padre.

Es importante destacar que de manera similar a las primeras versiones de EEs, estos algoritmos (PE) no implementan ningún operador de crossover como medio de exploración del espacio de búsqueda.

En cuanto al problema de la predicción de máquinas de estado finito, la definición original de PE se comportaba de la siguiente manera. Una población de padres (máquinas de estado finito), después de la inicialización, es expuesta a la secuencia de símbolos (el entorno) que ha sido observado hasta el momento actual. A medida que cada símbolo de entrada es presentado a cada máquina padre, se observa el correspondiente símbolo de salida (la predicción) y es comparado al próximo símbolo de entrada. Una función de pago predefinida es el medio para medir la capacidad de predicción de la máquina. Después que lo anterior es hecho, alguna función que considere la secuencia de pagos realizados es usada para indicar la calidad global de la máquina en cuestión. Una máquina descendiente (hijo) es creada a través de la mutación de la máquina padre. Por ejemplo, cambiar un símbolo de salida o cambiar una transición son ejemplos de operadores de mutación válidos para ser usados como operadores de mutación aplicados a máquinas finitas [61, 59, 60].

Luego que los hijos han sido creados usando los respectivos operadores de mutación sobre los miembros de la población, las máquinas que proveen los mejores pagos con respecto a una función de pago son retenidas con el objeto de conformar el conjunto de padres de la próxima generación. En general un único hijo es creado por cada padre, y la mitad del número total de máquinas es retenida de manera que se mantenga la población con un tamaño constante. Sin embargo, como en otros algoritmo evolutivos, es posible usar una población cuyo tamaño varíe a través del tiempo (generaciones).

Aunque PE fue propuesta en principio como un enfoque relacionado con inteligencia artificial, éste ha sido aplicado con diferentes grados de éxito a muchos problemas numéricos y de optimización combinatoria.

2.5.5. Algoritmos Genéticos

Los Algoritmos Genéticos (AGs) son los Algoritmos Evolutivos más conocidos y usados para la resolución de diversos problemas. John Ho-

lland [88] propuso el uso de algoritmos genéticos como una técnica de búsqueda eficiente para sistemas artificiales adaptativos. Debido a que los AGs están basados en la genética natural, el uso de un vocabulario acorde entre miembros de la comunidad es muy común. Los AGs trabajan sobre una población de individuos que son comúnmente denominados cromosomas o cadenas (strings). Un cromosoma es una secuencia de unidades o genes. El lugar en que cada gen está ubicado es llamado *loci* y su valor correspondiente *alelo*. Cada cromosoma representa, en general una posible solución a un problema dado. El operador de crossover es considerado el principal operador genético. Su definición original (Holland 1975 [88]) especifica que se toman dos cromosomas y se intercambia el material genético luego de una posición elegida aleatoriamente. El operador de mutación, generalmente considerado como un operador secundario en AGs, se define como cambios aleatorios en los valores de los genes en posiciones aleatorias y con una muy baja probabilidad. Sin embargo, el operador de crossover es considerado como la característica clave que diferencia a los AGs de los otros algoritmos evolutivos (un análisis detallado de estos y otros tópicos relacionados pueden ser consultados en [8, 9, 10]).

La estructura general de un Algoritmo Genético Simple (AGS) es la misma que la mostrada en la figura 2.11. Sin embargo, para un problema particular, las siguientes cinco componentes deben ser consideradas [114]:

- una representación para las soluciones potenciales al problema,
- una forma de crear la población inicial de soluciones potenciales,
- una función de evaluación que cumpla el papel del entorno, midiendo la calidad de las soluciones en término de su adaptabilidad (*fitness*),
- operadores genéticos que alteren la composición de los hijos,
- valores para varios parámetros que usa el algoritmo genético (tamaño de la población, probabilidades de aplicar los operadores genéti-

cos, etc.).

Desde los desarrollos iniciales de los AGs, un importante avance ha sido realizado permitiendo *evolucionar* las versiones iniciales de un AG a versiones mucho más avanzadas, muchas de ellas destinadas a la resolución de problemas complejos del mundo real. En esta parte de la descripción usaremos un AGS según [76, 114] para optimizar una función muy simple.

Tomando en cuenta las cinco principales componentes de un AG, consideramos el problema de maximizar la función objetivo $f(x) = x^2$. Por lo tanto es necesario decidir cuidadosamente acerca de la *representación*, *población inicial*, *función de fitness*, *operadores genéticos* y *los valores de los parámetros*.

- Representación.

Para esta función simple, asumimos que $x \in [0, 31]$, es decir, todos los posibles valores enteros para la variable x . La forma más natural de representar cada valor de este rango es a través de un string binario de longitud $l = 5$. Por lo tanto, para nuestro problema, un individuo podría visualizarse como:

$$C = b_4b_3b_2b_1b_0$$

donde $b_i \in \{0, 1\}$ para $i = 0, 1, 2, 3, 4$. El fenotipo para C es obtenido como $x = \sum_{i=0}^4 b_i 2^i$.

- Población Inicial.

Una alternativa muy simple es generar la población inicial de soluciones en forma aleatoria a través simular el tiro (no sesgado) de una moneda y asignar un valor del conjunto $\{0, 1\}$ a cada gen de cada cromosoma de la población.

■ Función de *Fitness*:

La medición del beneficio o utilidad de los individuos puede ser obtenida a partir del valor de la función objetivo, dado que se pretende maximizar la función objetivo⁵. Luego, mientras mayor sea el valor de la función objetivo, mayor será el valor de *fitness* asociado al individuo.

■ Operadores genéticos:

Un Algoritmo Genético Simple (AGS) consta de tres operadores genéticos [76]:

1. Selección: Este operador es una versión artificial de la selección natural y es usualmente llamado *operador de selección*. El objetivo de la selección es obtener una copia de individuos candidatos (padres) como el primer paso en el proceso de crear la próxima generación. El operador de selección puede ser implementado, por ejemplo, a través de la creación de una “Rueda de Ruleta” donde cada individuo corriente de la población tiene asociado una ranura en la rueda cuyo tamaño es proporcional a su respectivo *fitness*. Supongamos que f_i es el valor de *fitness* del individuo $i \in 1, \dots, \mu$, luego el respectivo tamaño de la ranura está dado por $p_i = \frac{f_i}{\sum_{j=0}^{\mu} f_j}$. El valor $p(i)$ representa la probabilidad de selección del individuo i , la cual es proporcional al *fitness* acumulado de la población actual. Cada vez que un individuo necesita ser seleccionado, se hace “girar” dicha rueda para saber cuál es el candidato. De esta manera, individuos con tamaño mayor de ranura, tendrán mayor probabilidad de ser seleccionados. El individuo seleccionado es luego incorporado en un conjunto de apareamiento, una población intermedia entre generación y generación, para luego aplicar los operadores genéticos de crossover y mutación (ver figura 2.12).

⁵Esto es posible si el rango de la función objetivo es un subconjunto de $\mathbb{R}^+ \cup \{0\}$.

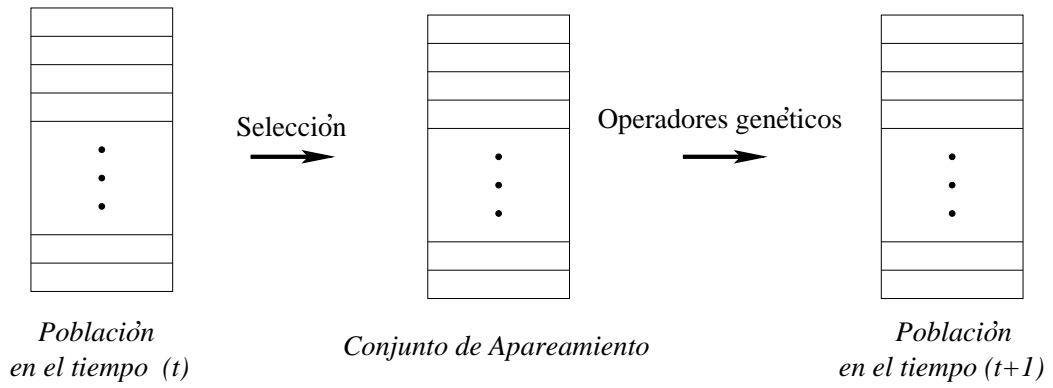


Figura 2.12: El conjunto de apareamiento como un paso previo a la aplicación de los operadores genéticos.

2. Crossover: Para cada par de cromosomas del conjunto de apareamiento, se genera un número aleatorio $pos \in [1 \dots l - 1]$. El número pos indica la posición del punto de cruce. Luego, dados dos cromosomas

$$A = (a_1 \dots a_{pos} a_{pos+1} \dots a_l)$$

$$B = (b_1 \dots b_{pos} b_{pos+1} \dots b_l)$$

los mismos son reemplazados por un par que representan a sus hijos respectivos:

$$A' = (a_1 \dots a_{pos} b_{pos+1} \dots b_l)$$

$$B' = (b_1 \dots b_{pos} a_{pos+1} \dots a_l)$$

Los hijos A' y B' heredan información genética de sus respectivos cromosomas padres, A y B . Este tipo de crossover es conocido con el nombre de *crossover de un punto*. Existen otros cuya variación está en el número de cortes que se realizan. Así tenemos el *crossover de dos puntos* y el *crossover uniforme*, entre otros.

3. Mutación: Es realizada un bit a la vez. Por ejemplo, para una representación clásica como la binaria, se realiza cambiando el valor del bit de 1 a 0 y viceversa.

Es importante destacar que no a todos los individuos del conjunto de apareamiento se les aplica los operadores de mutación y crossover. Por ejemplo, la aplicación del operador de crossover depende de una probabilidad p_c , es decir que un cierto número de hijos serán idénticos a sus padres. Similarmente ocurre con el operador de mutación, el cual se aplica con una probabilidad p_m .

- Parámetros: Sus valores dependen generalmente del problema al cual se aplica el algoritmo. Por ejemplo, para el caso del crossover, un operador primario en este contexto, existen valores pre-establecidos para p_c de alrededor de 0,6, mientras que para el operador de mutación, un operador secundario, la respectiva probabilidad de aplicación, p_m , tiene asociados por lo general valores muy pequeños (cerca de 0). De cualquier manera, esto no es una regla estricta. Además, poder asignar los valores correctos a cada uno de los parámetros de un AG (por ejemplo, p_c , p_m , tamaño de población, etc) es una tarea por demás ardua la cual ha sido y es un tema importante de investigación en la comunidad de computación evolutiva. En este sentido se han abierto líneas de investigación que incluyen entre otras, la adaptación automática de los valores de probabilidad de aplicación de ciertos parámetros teniendo en cuenta su aporte y rendimiento en distintas etapas de la búsqueda.

2.5.6. Programación Genética

La Programación Genética (PG) es una variante muy interesante de los AGs, con la diferencia de que ésta evoluciona estructuras jerárquicas dinámicas que describen, por lo general, programas de computadora,

[99, 100]. PG es implementada como un AE en el cual las estructuras de datos que participan en el proceso de evolución son programas ejecutables. El proceso de búsqueda de PG está dirigido a encontrar el mejor (o cercano al mejor) programa de computadora que al ser ejecutado producirá el mejor *fitness*, según el problema a resolver. La representación más frecuente en PG para los individuos es a través de una estructura de árbol y los operadores son diseñados de manera tal que preserven las restricciones sintácticas.

Es claro que podría ser posible manipular la población de individuos (árboles) usando los operadores estándar de un AGS, pero los hijos que sean generados de esta manera podrían no ser árboles dado que el operador de crossover es diseñado independientemente de una eventual interpretación del mismo. En consecuencia, lo que se busca es un operador de crossover que asegure que cada hijo generado sea sintácticamente correcto, es decir un árbol con las restricciones del lenguaje usadas para codificar un programa. De esta manera, vía un operador *ad hoc*, son eliminados del espacio de búsqueda programas sintácticamente inválidos, lo que conlleva una importante ventaja en el proceso de búsqueda. La figura 2.13 muestra la operación de crossover válida para una representación de árbol y que puede ser usada en PG. En este caso, un subárbol completo es identificado e intercambiado para crear un hijo. La simplicidad de la operación se debe principalmente a la naturaleza recursiva de la representación y por supuesto asumiendo que cada subárbol es sintácticamente correcto y que es ubicado correctamente en el árbol principal.

Una vez que una representación de árbol ha sido seleccionada, debemos considerar cómo esos árboles serán interpretados. En PG, las estructuras de árboles son consideradas como árboles de *parser* (derivación) de un lenguaje especificado por el usuario expresado en una sintaxis de Lisp. Este lenguaje es muy apropiado para PG dado que cada subárbol completo está delimitado por un conjunto de paréntesis perfectamente

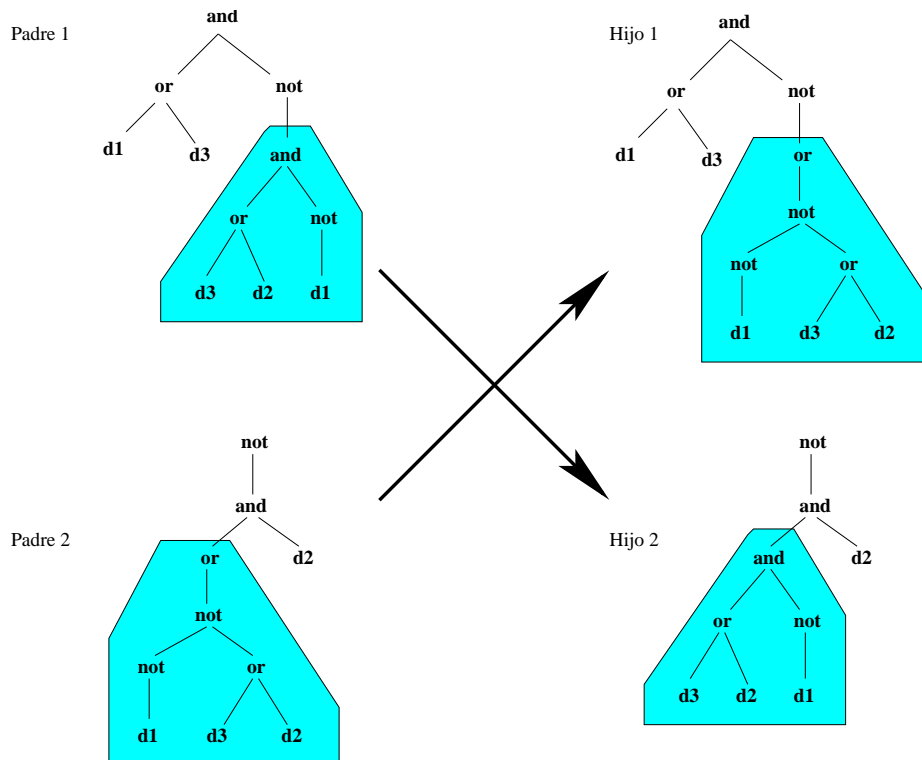


Figura 2.13: Crossover para representación de árboles en PG. Un subárbol de cada padre (resaltados en gris) es elegido en forma aleatoria e intercambiado entre los padres para crear dos hijos distintos. Este operador automáticamente preserva la sintaxis de la representación de árbol sin necesidad de un proceso de reparación posterior.

apareados. La aplicación del crossover sobre una expresión en lenguaje Lisp es simplemente una cuestión de intercambiar completamente listas entre los padres y la operación que es común a los dos y que es fácilmente codificada en programas escritos en Lisp.

Los lenguajes en programación genética incluyen funciones que toman argumentos y éstas aparecen en los nodos internos del árbol, mientras que las hojas del árbol representan los datos respectivos. Es importante notar que Lisp ha sido uno de los lenguajes más escogidos para la implementación en las cuales los programas son tratados como datos. En este caso, los programas son datos mientras éstos están siendo usados como material genético por los respectivos operadores, pero son considerados programas ejecutables cuando ellos son evaluados para determinar su *fitness*. A medida que PG fue evolucionando en Lisp, los programas

que se iban ejecutando se parecían cada vez menos a Lisp. Es decir, los programas (cromosomas) se estructuraban como árboles como en Lisp, pero muchas, sino casi todas, de las funciones usadas en los programas evolucionados no correspondían a funciones Lisp (ver [9], capítulo 11). Alrededor de 1992, muchos investigadores implementaron sistemas del tipo PG en lenguaje C y C++, entre otros lenguajes de programación.

La programación genética se define muchas veces como una variante de AGs donde la diferencia fundamental está puesta en la representación, una selección apropiada de operadores genéticos, y la función de *fitness* de la representación, que implica la ejecución del programa (cromosoma) en una máquina virtual. Si se toma PG en forma aislada, estos elementos no capturan la esencia de PG para diferenciarlo de AGs. Lo que es interesante diferenciar es que los AGs evolucionan cromosomas que representan una solución para un determinado problema, mientras que PG evoluciona programas de computadoras que permiten resolver un problema.

2.5.7. AEs: algunos conceptos importantes

En esta sección se describen algunos de los conceptos más destacados en relación a las distintas componentes de un algoritmo evolutivo. En las secciones previas se establecieron algunas diferencias entre los distintos enfoques evolutivos, aunque todos comparten, de una u otra manera, ciertos conceptos que eventualmente podrían ser usados en cualquiera de los enfoques. En lo que resta de los capítulos de la presente tesis y con el objetivo de establecer una nomenclatura homogénea, nos referiremos al término general Algoritmo Evolutivo (AE) a una versión mejorada o tal vez más sofisticada de lo que usualmente se denomina Algoritmo Genético canónico (su versión original). Es decir que en este contexto un AE es un AG con características más avanzadas con respecto a: representación, métodos de selección, operadores de búsqueda, manejo de restricciones, etc.

La idea esencial de la resolución de un problema a través de un enfoque evolutivo es muy simple. Se tiene una población de soluciones candidatas las cuales son evolucionadas en sucesivas iteraciones a través de operadores especializados. Estos operadores producen ciertas variaciones en las soluciones de la población lo que permite la exploración del espacio de búsqueda. La selección determina el conjunto de soluciones que deberían mantenerse en la población para ser usadas como base para el siguiente paso de exploración del algoritmo. Como consecuencia de esta repetición de pasos, se busca como meta alcanzar una solución o conjunto de soluciones que satisfagan determinadas condiciones de calidad considerando el problema que se pretende resolver. Una característica deseable de este proceso es que sea rápido. Sin embargo, la velocidad con la cual puedan ser alcanzadas soluciones adecuadas, es en parte, dependiente de las elecciones que se realizan en cuanto a la representación de las soluciones, función de evaluación, operadores específicos de exploración y selección, tamaño de la población y el procedimiento usado para crear una población inicial, entre muchos otros que dependen de las características del problema.

2.5.7.1. Representación

Todos los problemas de optimización y búsqueda deben tener en cuenta soluciones. La estructura de una solución en general depende del problema subyacente. En algunos problemas, las soluciones son representadas a través de vectores de números reales los cuales satisfacen ciertas restricciones de los parámetros del problemas. En otros problemas, las soluciones representan una estrategia o un “algoritmo” para alcanzar una determinada tarea. En síntesis, la estructura de una solución varía de un problema a otro y al mismo tiempo un problema puede aceptar más de una estructura para representar una posible solución. En general, un método de búsqueda es más eficiente cuando usa un determinado tipo de representación y no algún otro de entre las posibles alternativas. Por

lo tanto, la representación escogida no sólo puede llegar a depender del problema a resolver, sino también del tipo de método usado. En síntesis, la eficiencia y complejidad del algoritmo de búsqueda depende en gran medida de cómo las soluciones han sido representadas y cuán adecuada es la representación en el contexto del problema a resolver. En algunos casos, un problema difícil puede tornarse más sencillo si se elige una representación que trabaje eficientemente con un algoritmo particular.

Las representaciones más comunes que aparecen en la literatura de AEs incluyen: cadenas binarias, vectores de números reales, permutaciones, vectores de números enteros, árboles de sintaxis y muchas otras [114, 9, 10].

2.5.7.2. Métodos de selección

La selección es uno de los principales operadores en AEs. El objetivo primario de este operador es elegir de la población, soluciones relativamente buenas, pero no crea nuevas soluciones. Este proceso implica eliminar de la población aquellas que no son de buena calidad. En consecuencia, el operador de selección es una mezcla de dos conceptos: reproducción y selección. Esto es, cuando una o más copias de una solución es obtenida, se denomina reproducción. De este conjunto de soluciones obtenidas por copias de soluciones existentes en la población, se realiza la selección de las soluciones superiores para recién aplicar alguno de los operadores de exploración. Muchos estudios en AEs usan esta división de conceptos, mientras que otros, solamente lo denominan selección.

La identificación de la calidad de las soluciones es usualmente llevada a cabo a través del *fitness* de la solución. La idea esencial es que una solución que tiene un mejor valor de *fitness*, debería tener una mayor probabilidad de sobrevivir, es decir, de ser seleccionada. Sin embargo, existe una serie de variaciones en cuanto a la forma en que un operador o método de selección lleva a cabo esta tarea.

Algunos operadores ordenan la población según los valores de *fitness* y

determinísticamente eligen las que se encuentran en los primeros lugares del ordenamiento, mientras que otros operadores asignan una probabilidad de selección a cada solución proporcional a su contribución al *fitness* total de la población y producen las copias de las soluciones de acuerdo a la distribución de probabilidad obtenida.

En lo que sigue, se describen brevemente algunos de los métodos más conocidos de selección: *proporcional*, *torneo*, *ranking*, *Boltzmann* (estocásticos) y por otro lado, $(\mu + \lambda)$ y (μ, λ) , ambos métodos determinísticos.

En la selección proporcional [76], el número esperado de copias que recibe una solución es proporcional a su *fitness*. De esta manera, una solución que tenga asignado un valor de *fitness* que sea el doble del asignado a una segunda solución, tendrá como consecuencia que la primera reciba tanto como dos veces el número de copias que la segunda (esto es en términos del valor esperado). La forma más sencilla de implementar este método es a través del esquema conocido como *Rueda de la Ruleta*. En este esquema, cada solución tiene asignado un sector en la “rueda” cuyo tamaño es proporcional a su *fitness*. Para realizar la selección de una solución, se hace girar la rueda (conceptualmente hablando) tantas veces como lo indique el tamaño de la población y cada vez seleccionando la solución que indica la aguja de la rueda luego de detenerse. Dado que el tamaño de cada sector de la rueda es proporcional al *fitness* de la solución que representa, una solución con un valor de *fitness* alto, recibirá con mayor probabilidad, un número mayor de copias que una solución con un menor valor de *fitness*. Este método, si bien puede ser aplicado en muchos problemas, tiene algunas limitaciones importantes. No acepta valores negativos de *fitness*; no puede manejar problemas de minimización directamente; pueden aparecer super-individuos al principio de la ejecución, lo que podría llevar al algoritmo a converger prematuramente a valores sub-óptimos. Algunos de estos problemas pueden ser resueltos usando algún tipo de escalamiento de la función de *fitness*, ya sea para

transformar todos los valores posibles a valores positivos o bien para disminuir las diferencias relativas entre buenas y malas soluciones, en este último caso, para evitar la convergencia prematura [76].

En la selección por torneo no es necesario realizar ningún tipo de escalamiento como en la selección proporcional, lo que no implica que no pueda producirse convergencia prematura, dado que esta situación puede depender de varios factores. En este método, se realiza un “torneo” en el que participan q soluciones elegidas en forma aleatoria de la población. Una solución, de las q soluciones, puede ser escogida determinísticamente o probabilísticamente. Una vez que el torneo ha sido jugado, dos opciones pueden aplicarse: a) las soluciones vuelven a la población para participar en el próximo torneo o bien, b) se mantienen hasta que se han jugado un número determinado de torneos. Independientemente de estas alternativas y posibles valores para q , en general se usa una de las forma más simples denominada *selección por torneo binario*. Esta forma de selección escoge dos soluciones en forma aleatoria de la población y aquella con mejor valor de la función objetivo es la seleccionada. La ventaja del método de selección por torneo es puede que manejar problemas de minimización o maximización en forma indistinta dado que no requiere cambios en la función de *fitness* (de hecho no la necesita). Además no existe ningún tipo de restricción al rango de la función objetivo, ya que éste puede ser negativo, positivo o ambos. Otra de sus ventajas es la posibilidad de realizar implementaciones paralelas dado que cada torneo es un proceso que se realiza en forma independiente de los otros torneos ([8], C2.3).

El método de selección por *ranking* es similar al proporcional, excepto que las soluciones son ordenadas en forma descendente (minimización) o ascendente (maximización) según los valores de la función objetivo. A cada solución se le asigna un valor de *fitness* acorde a la posición en el ranking. Es importante destacar que los valores de *fitness* para cada posición es calculado previamente al ordenamiento. Existen distintas

maneras de hacer dicha asociación de valores y por cada una de ellas existen parámetros que determinan los valores relativos de *fitness* por cada posición. En otras palabras, este método usa la rueda de la ruleta, pero el tamaño de cada sector es obtenido de una manera independiente del valor de *fitness* de las soluciones. El ordenamiento se realiza para asociar los valores de *fitness* previamente calculados con la posición que ocupe cada solución en dicho ordenamiento ([8], C2.4).

El método de selección de *Boltzmann* trabaja asignando un *fitness* que depende de la siguiente distribución de probabilidades: $\mathcal{F}_i = 1/(1 + \exp(f_i/T))$, donde f_i es el valor de la función objetivo aplicada a la solución i y T es un parámetro análogo a la temperatura en términos de la distribución de Boltzmann. El valor de este parámetro es reducido de una manera predeterminada a medida que las iteraciones o generaciones avanzan. El parámetro T es inicializado con un valor grande. En ese momento, la distribución de probabilidades se asemeja a una uniforme, es decir, todas las soluciones son equiprobables. A medida que la ejecución avanza, el valor de T disminuye y sólo las mejores soluciones serán seleccionadas. Autores como T. Bäck sostienen que ésta es simplemente otra manera de realizar un escalamiento en la selección proporcional ([9], Cap.27).

En la selección $(\mu + \lambda)$, λ hijos son creados a partir de μ padres. Luego un conjunto de $(\mu + \lambda)$ es obtenido y de este conjunto los μ mejores individuos son seleccionados. El otro método determinístico, similar al anterior, es (μ, λ) en el cual los nuevos μ mejores individuos son obtenidos de un conjunto de $\lambda \geq \mu$ creados a partir de los μ originales.

Cualquier que sea el esquema de selección usado en un algoritmo evolutivo, éste involucra dos elementos: a) un conjunto de selección y b) una distribución de probabilidades asociadas a este conjunto. El conjunto de selección es obtenido para la reproducción (copias) y la eliminación de individuos. Un punto importante a clarificar es ¿qué es lo que contiene dicho conjunto? En la fase de reproducción, los padres son seleccionados

para producir los hijos; en ese caso el conjunto de selección contiene la población actual. En la fase de eliminación (aquellos no seleccionados para producir hijos) se debe decidir acerca de los individuos que deberían ser eliminados para hacer lugar a la nueva generación (asumiendo un tamaño fijo para la población). Por ejemplo, los AGs aplican políticas de reemplazo generacional introducidas por Holland [88]. La primera política, denominada R_1 , establece que en cada iteración es creado solamente un individuo hijo. Este nuevo individuo deberá ser insertado en la población actual y, por lo tanto, es necesario eliminar alguno de dicha población para dar lugar al individuo recientemente creado. La segunda política, denominada R_d , establece que todos los individuos de la población que fueron seleccionados, según el número esperado de hijos, darán lugar a una población la cual reemplazará completamente a la población actual luego de aplicado el proceso de recombinación. El primer intento de evaluar empíricamente las propiedades de R_1 y R_d fue llevado a cabo por De Jong [94]. En este estudio se usó un parámetro G , llamado *brecha generacional*, el cual se definió para introducir el concepto de poblaciones superpuestas. El parámetro G controla la parte de la población que será reemplazada en cada generación. Así, $G = 1$ indica reemplazo total de la población, es decir la política R_d y $G = 1/m$ (m es el tamaño de la población) corresponde a la política R_1 (reemplazo de un único individuo).

Whitley [174] y Whitey et al. [175] propusieron una alternativa generacional llamada *steady state* en la cual padres e hijos coexisten. Los AEs de tipo *steady state* se caracterizan por el hecho de que uno o dos hijos son producidos en cada generación. El conjunto de selección para la eliminación puede consistir solamente de los padres o bien ser extendida con los hijos producidos. Claramente es necesario determinar una política para hacer lugar a los hijos recientemente creados. En el otro extremo se encuentran los sistemas generacionales en donde la población completa es reemplazada por la población de hijos recientemente creada.

En este caso el tiempo de vida de cada individuo está limitado a una generación. De esta manera, steady state implementa un sistema de poblaciones superpuestas en los cuales se espera que el tiempo de vida de un individuo pueda ser mayor a una generación. En los sistemas generacionales las poblaciones no son superpuestas. Adicionalmente, se pueden usar estrategias elitistas para asegurar la subsistencia de individuos de buena calidad más allá de una generación.

2.5.7.3. Operadores

Un algoritmo evolutivo procesa una población de individuos $P(t) = \{a_1^t, \dots, a_n^t\}$ donde t es el número de iteración y cada individuo representa una solución posible del problema a resolver. Tal como se mencionó previamente, existen varias alternativas para representar un individuo. Dichas representaciones pueden variar desde simples cadenas binarias a estructuras complejas (por ejemplo, un árbol). Algunas soluciones de $P(t)$ son seleccionadas (párrafo anterior) para conformar un conjunto de soluciones que darán lugar a nuevos individuos. Para llegar a estos nuevos individuos hijos se realiza una transformación sobre los individuos padres a través de uno o más operadores (denominados genéticos, de exploración o de búsqueda). Los dos operadores básicos de un AE son la mutación (unario) y el crossover (binario⁶). El operador de mutación (m) crea un nuevo individuo produciendo, en general, pequeños cambios en el individuo original. El crossover o recombinación produce uno o varios hijos intercambiando partes de las soluciones que representan los individuos padres. En resumen, la representación usada para los individuos y los operadores genéticos son una de las partes más importantes en el diseño de un algoritmo evolutivo y ciertamente, influyen directamente o indirectamente en la eficacia del mismo para resolver un determinado problema. Es importante destacar que para cada representación posible pueden existir una gran cantidad de posibilidades en cuanto al diseño de

⁶En sus versiones más usuales, aunque no está limitado a dos operandos

operadores genéticos. Esta situación será tratada en detalle en el capítulo 4 en el contexto de manejo de restricciones usando metaheurísticas.

2.5.7.4. Valor de *Fitness*

La obtención de un valor de *fitness* depende de la codificación usada para representar las soluciones. Esta codificación depende muchas veces del tipo de algoritmo usado (AGs, EEs, PE, o PG) aunque no siempre es tan estricto. Cuando nos referimos a AGs, es posible pensar en que los valores de *fitness* asociados a un individuo requieren una decodificación, esto es, un mapeo de la representación del individuo o cromosoma a un valor o conjunto de valores que representen el dominio o parámetros del problema en sí. Por ejemplo, si un individuo es representado por una cadena de 0's y 1's, y el problema tiene un parámetro que es un valor entero, luego para obtener el valor de *fitness* será necesario obtener el entero respectivo de la secuencia de bits. En este sentido, decimos que el algoritmo manipula (vía crossover y mutación) una representación codificada de los parámetros del problema y no los parámetros en sí mismos. Más formalmente, supongamos que un problema de optimización se define a través de la siguiente función objetivo:

$$f : M \Leftrightarrow \mathbb{R}$$

donde M representa el espacio de búsqueda de la función f . Luego, la función de evaluación o valor de *fitness* F es descrito como:

$$F : R \xrightarrow{d} M \xrightarrow{f} \mathbb{R} \xrightarrow{s} \mathbb{R}_+$$

$$F = s \circ f \circ d$$

donde R es el espacio de la representación del cromosoma, d es la función de decodificación, y s es una función de escalamiento⁷ la cual es usada en combinación con el método de selección proporcional.

⁷Debe tenerse en cuenta que no siempre es necesario la aplicación de una función de escalamiento. Esto dependerá básicamente del método de selección utilizado.

Por ejemplo, cuando se pretende codificar un espacio n -dimensional de valores reales para una función objetivo f_n usando una cadena de bits, la ley de conformación de la función de *fitness* F podría ser la siguiente:

$$F : \{0, 1\}^l \xrightarrow{d_b} \mathbb{R}^n \xrightarrow{f_n} \mathbb{R} \xrightarrow{s} \mathbb{R}_+$$

donde l es la longitud de los cromosomas y d_b es la codificación binaria de un vector real. Así, d_b mapea segmentos del cromosoma en números reales.

El análisis previo es relativamente general, ya que un cromosoma podría ser representado directamente por un vector de reales, y en este sentido al algoritmo manipula los parámetros del problema en forma directa y no necesitaría realizar ningún tipo de operación de decodificación. Esto es una pequeña muestra de las posibles variaciones que se pueden presentar en el diseño de un algoritmo evolutivo.

Otro ejemplo es un problema que involucra un cierto número de variables de decisión. En este caso, el esquema de codificación que se ajusta naturalmente es a través de cadenas binarias. Sin embargo, aquí no es necesario realizar ningún tipo de decodificación para obtener el valor de *fitness* del individuo ya que éste en sí mismo representa en forma directa una posible solución al problema.

$$F : \{0, 1\}^l \xrightarrow{f} \mathbb{R} \xrightarrow{s} \mathbb{R}_+$$

2.5.8. Aplicaciones de AEs

El área de computación evolutiva es una de las más prolíficas en cuanto a la variedad y campos de aplicación de los algoritmos desarrollados hasta el momento. Esto está evidenciado no sólo por el creciente número de publicaciones relacionadas con este tema, sino también por el creciente interés de diversas comunidades de investigadores interesados en usar estos métodos para resolver problemas de sus respectivas áreas para los que no es fácil (y, a veces es, incluso, imposible) diseñar algoritmos

eficientes y/o exactos. Las áreas de aplicación de AEs es muy amplia e incluye: planificación, diseño, control, redes neuronales, optimización combinatoria y numérica, problemas restringidos, robótica, procesamiento de imágenes y señales digitales, arte digital, y muchas otras.

Capítulo 3

Metaheurística ACO

El presente capítulo, el cual es continuación del anterior, presenta a la otra metaheurística basada en una población estudiada en esta tesis. Dado que la aparición de esta metaheurística es relativamente reciente y que nuestra propuesta fue motivada principalmente por la posibilidad de incorporarle el enfoque de frontera, hemos dedicado un capítulo aparte a su descripción.

3.1. Optimización en Base a Colonias de Hormigas (ACO)

Los algoritmos basados en colonia de hormigas (AH¹) son sistemas multi-agentes en los cuales el comportamiento de una única hormiga, llamado *hormiga artificial* o simplemente *hormiga*, está inspirado por el comportamiento de hormigas reales. Los AHs son uno de los ejemplos más exitosos de sistemas de inteligencia colectiva, los cuales han sido aplicados a muchos tipos de problemas, desde el problema de Viajante de Comercio a Ruteo en redes de telecomunicaciones [26, 39].

¹El término usado en Inglés es *ant algorithms* y de aquí en más usaremos la sigla AH para referenciarlos cuando sea necesario

3.1.1. Introducción

El desarrollo de los AHs está inspirado en experiencias de laboratorio realizadas por Goss et al. [77] usando colonias de hormigas reales. Las hormigas que conforman la colonia de laboratorio estaban compuestas de hormigas originarias de Argentina *Irimidex humilis*, las cuales tenían acceso a una fuente de alimentación en un terreno en el cual el nido de la colonia estaba conectado por un puente con dos brazos de diferente longitud (ver Figura 3.1). Los brazos del puente son ubicados de manera tal que las hormigas que van en cualquier dirección (desde el nido hacia la fuente de alimento o viceversa) deben elegir entre uno u otro. De la observación del experimento se deduce que después de varios minutos (fase transitoria) la mayoría de las hormigas usa el camino más corto. Se observa además, que la probabilidad de seleccionar el camino más corto, se incrementa con la diferencia de la longitud entre los dos brazos del puente. La emergencia del comportamiento de selección del paso más corto puede ser explicada en términos de *autocatálisis* (*retroalimentación positiva*) y la *longitud de paso diferencial*, hecho que es posible debido a una forma de comunicación indirecta, conocida como *stimergy*, que consiste básicamente en modificaciones locales del entorno. Más es-

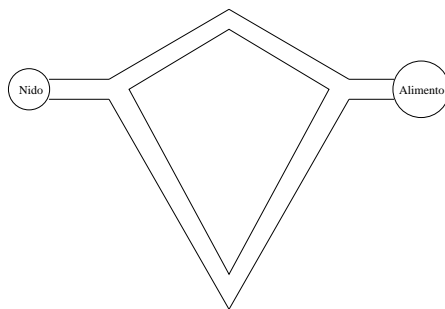


Figura 3.1: Puente con dos brazos para la realización del experimento.

pecíficamente, las hormigas, mientras van desde el nido a la fuente de alimento y viceversa, depositan sobre la superficie una sustancia química llamada *feromona*. Al momento de llegar al punto de decisión, donde el puente se abre en dos brazos, realizan una elección probabilística ses-

gada por la intensidad de feromona que perciben sobre los dos brazos. El comportamiento tiene un efecto auto-catalítico debido al hecho de que la elección del paso incrementará la probabilidad de que dicho paso sea elegido por otras hormigas en el futuro. Es claro que al comienzo del experimento no existe feromona en ninguno de los dos brazos y por lo tanto, en el comienzo, las hormigas que van desde el nido a la fuente de alimento elegirán con igual probabilidad cualquiera de ellos. Pero debido a la longitud diferencial del paso, las hormigas que eligieron el paso más corto llegarán primero a la fuente de alimento. Luego, de vuelta al nido, alcanzan nuevamente el punto de decisión y, con una alta probabilidad, encontrarán más feromona sobre el paso más corto. Dicha feromona fue liberada por aquellas hormigas que eligieron el paso más corto en su camino hacia la fuente de alimento. En el camino de regreso, más feromona es liberada sobre el paso elegido, lo que lo hará más atractivo para futuras decisiones realizadas por las hormigas de la colonia. A medida que el proceso itera, la cantidad de feromona depositada sobre el paso más corto tenderá a crecer más rápido que aquella depositada sobre el paso más largo, lo cual llevará a que más y más hormigas lo elijan en detrimento del paso más largo.

3.1.2. La metaheurística ACO

El algoritmo más simple en la clase de algoritmo definido por la metaheurística ACO es el S-ACO, el cual implementa el comportamiento básico de una colonia de hormigas. S-ACO puede ser usado para encontrar una solución al problema del paso más corto definido sobre un grafo $G = (V, E)$ con $|V| = n$, donde una solución es un camino en el grafo que conecta un nodo fuente s (nido) con un destino d (la fuente de alimento) tal como lo muestra la Figura 3.2. Los experimentos iniciales usando un simple grafo para modelar el aparato de la Figura 3.1 ha demostrado que S-ACO encuentra en forma efectiva el camino más corto entre el nido y la fuente de alimento simulados. Sin embargo, cuando la complejidad del

aparato se incrementa, el comportamiento de S-ACO tiende a ser menos estable. De cualquier manera, a partir de S-ACO han sido desarrolladas diversas variantes de los algoritmos ACO que han permitido mejorar los resultados obtenidos para éste y una importante cantidad de problemas de optimización, particularmente aquellos cuya solución puede ser representada como una secuencia de nodos en un grafo². Dichos algoritmos pertenecen a la clase de algoritmos definidos según la metaheurística ACO. En general, los algoritmos ACO pueden ser aplicados a problema de optimización discretos que pueden ser categorizados de la siguiente manera:

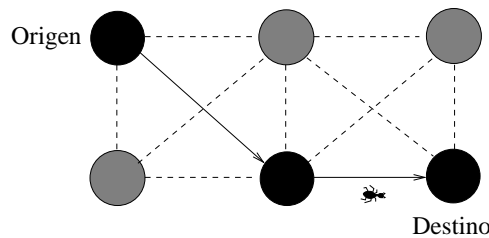


Figura 3.2: La construcción de una solución implica recorrer el grafo desde un nodo *origen* hasta alcanzar un nodo *destino*. Aquellas hormigas que sigan la línea llena, llegarán primero al nodo destino y por ende dirigirán la búsqueda hacia este paso cuando vuelvan hacia el nodo origen.

- Un conjunto finito de *componentes* $C = \{c_1, c_2, \dots, c_n\}$.
- Un conjunto finito L conformado por las posibles *conexiones/transiciones* entre los elementos de C definidos sobre un subconjunto \tilde{C} del producto cartesiano $C \times C$, $L = \{l_{c_i c_j} | (c_i, c_j) \in \tilde{C}\}$, $|L| \leq n^2$.
- Para cada $l_{c_i c_j} \in L$ una función *costo de conexión* $J_{c_i c_j}$, posiblemente parametrizada según una medida de tiempo t .
- Un conjunto finito de *restricciones* $\Omega \equiv \Omega(C, L, t)$ asociadas a los elementos de C y L .

²Debe tenerse en cuenta que esto corresponde a la formulación original del método y por lo tanto no es excluyente. En el capítulo 5 se verá el método desde otra posible perspectiva.

- Los estados del problema son definidos en términos de las secuencias $s = \langle c_i, c_j, \dots, c_k, \dots \rangle$ sobre los elementos de C (equivalentemente de L). Si S es el conjunto de todas las posibles secuencias de longitud a lo sumo n , luego el conjunto \tilde{S} de todas las subsecuencias que son factibles con respecto a las restricciones $\Omega = (C, L, t)$, es un subconjunto de S . Los elementos en \tilde{S} definen los *estados factibles* del problema. La longitud de una secuencia s , esto es, el número de componentes en la secuencia, es expresado por $|s|$.
- Una estructura de *vecindad* la cual es definida como sigue: el estado s_2 es el vecino de s_1 si (i) ambos s_1 y s_2 son elementos de \tilde{S} , (ii) el estado s_2 puede ser alcanzado desde s_1 en un paso lógico, es decir, si c_1 es la última componente en la secuencia que determina el estado s_1 , debe existir $c_2 \in C$ de manera tal que $l_{c_1 c_2} \in L$ y $s_2 = s_1 \oplus \langle c_2 \rangle^3$.
- Una *solución* ψ es un elemento de \tilde{S} que satisface todas los requerimientos del problema. Una solución se denomina *multi-dimensional* si está definida en términos de múltiples secuencias distintas conformadas por elementos de C .
- Un costo $J_\psi(L, t)$ es asociado a cada solución ψ . $J_\psi(L, t)$ es una función de todos los costos $J_{c_i c_j}$ de todas las conexiones que pertenecen a la solución.

Consideremos un grafo $G = (C, L)$ asociado a una instancia dada de un problema de optimización discreto según se describió previamente. Una solución al problema de optimización puede ser expresado en términos de pasos factibles sobre el grafo G . Los algoritmos ACO, en su concepción original, son directamente aplicables para encontrar en un grafo, pasos factibles (secuencias) de costo mínimo con respecto a un conjunto de restricciones Ω .⁴

³ $\oplus : S \times S \rightarrow S$ da como resultado una secuencia formada por las dos secuencias dadas como parámetros.

⁴por ejemplo, en el Problema del Viajante de Comercio, definido en la sección 3.1.3, C es el conjunto de ciudades, L es el conjunto de arcos que conectan las ciudades, y una solución ψ es un

En los algoritmos ACO, una población (colonia) de agentes (hormigas) resuelven un problema de optimización en forma colectiva a través de una representación de grafo. La información recolectada por las hormigas durante el proceso de búsqueda es codificada en *rastros de feromona* τ_{ij} asociados a la conexión l_{ij} ⁵. El rastro de feromona codifica una memoria a largo término acerca de proceso de búsqueda global. Por lo tanto, dependiendo de la representación del problema considerada, el rastro de feromona puede estar asociado a todos a algunos de los arcos que conectan las componentes del problema. Los arcos pueden también tener asociado un *valor heurístico* η_{ij} que representa el conocimiento de una instancia del problema o bien, información provista por una fuente diferente a las hormigas que conforman el sistema.

En suma, una colonia de hormigas presenta las siguientes características generales:

- Aunque cada hormiga, en forma aislada, es capaz de encontrar una solución (probablemente pobre) al problema considerado, soluciones de buena calidad solamente pueden emerger como el resultado de la interacción colectiva entre las hormigas de la colonia.
- Cada hormiga hace uso solamente de información privada y local al nodo que está visitando⁶.
- Las hormigas se comunican entre ellas indirectamente a través de la información que “leen” o “escriben” en las variables que representan los rastros de feromona acumulados.
- Las hormigas no son adaptativas en sí mismas, por el contrario, ellas, en forma adaptativa modifican la manera en que el problema es representado y percibido por el resto de las hormigas en la colonia (*stigmergy*).

circuito Hamiltoniano.

⁵De aquí en más se simplifica la notación haciendo $l_{c_i c_j} = l_{ij}$.

⁶En lo siguiente, los términos nodo y componente, como también arco y conexión, serán usados indistintamente a través de esta tesis.

Por otro lado, las hormigas de una colonia tienen las siguientes propiedades:

- Una hormiga busca soluciones factibles de costo mínimo

$$\hat{J}_\psi = \min_\psi J_\psi(L, t).$$

- Una hormiga k tiene una memoria asociada \mathcal{M}^k que es usada para almacenar información sobre el paso recorrido hasta el momento. Dicha memoria puede, por ejemplo, ser usada para (i) construir soluciones factibles, (ii) evaluar la solución encontrada y/o (iii) volver sobre el paso para modificar los niveles de feromona.
- Una hormiga k en el estado $s_r = \langle s_{r-1}, i \rangle$ puede moverse a cualquier nodo j que se encuentre en el *vecindario factible* \mathcal{N}_i^k , definido como $\mathcal{N}_i^k = \{j | (j \in \mathcal{N}_i) \wedge (\langle s_r, j \rangle \in \tilde{S})\}$.
- Una hormiga k puede ser asignada a un *estado inicial* s_s^k y una o más condiciones de terminación e^k ⁷.
- Las hormigas comienzan desde el estado inicial y se mueven hacia estados vecinos factibles, construyendo las soluciones en forma incremental. El proceso de construcción para una hormiga k se detiene cuando al menos una de las condiciones de terminación e^k es satisfecha.
- Una hormiga k ubicada sobre el nodo i puede moverse al nodo $j \in \mathcal{N}_i^k$. El movimiento es seleccionado a través de la aplicación de una regla de decisión probabilística.
- La regla de decisión probabilística es una función de (i) los valores almacenados en una estructura de datos local al nodo $\mathcal{A}_i = [a_{ij}]$

⁷Usualmente, el estado inicial es expresado como una secuencia de longitud uno, es decir, con una única componente.

llamada *tabla de ruteo*, la cual es obtenida a través de una composición funcional del rastro de feromona y valores heurísticos disponibles localmente, (ii) la memoria privada de cada hormiga la cual almacena su historia pasada en relación a los nodos visitados, y (iii) las restricciones del problema.

- Cuando se produce un movimiento desde el nodo i al nodo vecino j , la hormiga puede modificar el rastro de feromona τ_{ij} sobre el arco (i, j) . Este tipo de modificación es denominada *modificación de feromona paso-a-paso* (conocida y referida como *on-line*).
- Una vez que una solución es construida, la hormiga puede recorrer el camino andado en forma inversa y modificar los rastros de feromona en todos los arcos recorridos. Este tipo de modificación es denominada *modificación de feromona demorada* (conocida y referida como *on-line*).
- Una vez que cada hormiga ha construido una solución, y que eventualmente realicen el recorrido inverso hasta el nido, decimos que las hormigas mueren y liberan todos los recursos que han ocupado. Aunque podríamos decir que liberan los recursos y quedan listas para iniciar nuevamente el proceso de recorrido del grafo hacia la fuente de alimento.

El comportamiento de las hormigas en un algoritmo ACO puede ser resumido de la siguiente manera: Una colonia de hormigas se mueve concurrentemente y asincrónicamente a través de los estados adyacentes del problema haciendo movimientos a través de los nodos vecinos del grafo G . Ellas se mueven aplicando una política de decisión local estocástica que hace uso de la información contenida en las tabla de ruteo locales a cada nodo. A través de dichos movimientos, las hormigas construyen las soluciones al problema en forma incremental. Una vez que una hormiga ha construido una solución, o mientras la solución está siendo construida, la hormiga evalúa la solución (parcial) y deposita información según

su calidad, como rastros de feromona sobre las conexiones usadas (transitadas). De esta manera, la información representada por los valores de feromona, dirigirá la búsqueda de las hormigas en futuros ciclos del algoritmo.

Además de las actividades propias de las hormigas, un algoritmo ACO incluye un par de procedimientos adicionales: *evaporación del rastro de feromona* y *acciones auxiliares*⁸, esta última opcional. La evaporación de la feromona es un procedimiento a través del cual se disminuye la intensidad del rastro de feromona sobre las conexiones a medida el tiempo avanza. Desde el punto de vista práctico, la evaporación de la feromona es necesaria para evitar una convergencia prematura del algoritmo dirigida a una región subóptima del espacio de búsqueda. La evaporación implementa un suerte de “olvido” de las conexiones visitadas a fin de favorecer la exploración de nuevas áreas del espacio de búsqueda. Las acciones auxiliares pueden ser usadas para implementar acciones centralizadas las cuales no podrían ser realizadas por las hormigas individualmente. Por ejemplo, la activación de un procedimiento de búsqueda local, o la colección de información global que pueda ser usada para decidir si es útil o no depositar feromona adicional para sesgar el proceso de búsqueda desde una perspectiva no-local. En este último caso, el demonio podría usar la mejor solución encontrada para reforzar los valores de feromona sólo sobre ciertas conexiones (aquellas que conforman la mejor solución encontrada). Este es un clásico ejemplo de modificación *off-line* de los valores de feromona.

3.1.3. Un algoritmo ACO para el Problema del Viajante de Comercio

En esta sección se presenta un algoritmo ACO aplicado al Problema del Viajante de Comercio (TSP). El TSP juega un rol muy importante

⁸El término original en Inglés es *daemon*, concepto que caracteriza a procesos que se ejecutan cuando se cumplen determinadas condiciones.

en ámbito de los algoritmos ACO dado que fue el primer problema con el cual se experimentó con las primeras versiones de los algoritmos ACO (Dorigo et al [26, 39, 50]). El TSP fue elegido por varias razones: (i) es un problema para el cual la metáfora de la colonia de hormigas puede ser aplicado muy fácilmente, (ii) es uno de los problemas \mathcal{NP} -Completo más estudiados en optimización combinatoria [68], y (iii) su formulación es muy sencilla, de manera tal que el comportamiento del algoritmo no se ve oscurecido por cuestiones complejas del problema.

El TSP, de acuerdo a la terminología introducida en la sección previa, puede ser definido como sigue. Sea C el conjunto de componentes o ciudades del problema. Sea L el conjunto de conexiones conectando totalmente los elementos de C , y $J_{c_i c_j}$ el costo asociado a la conexión entre la ciudad C_i y C_j , esto es, la distancia entre las ciudades i y j . El problema TSP consiste en encontrar un circuito Hamiltoniano de longitud mínima sobre el grafo $G = (C, L)$. Un circuito Hamiltoniano del grafo G es un tour cerrado ψ que incluye todos los n nodos de G pero sin repetir ninguno. La longitud del tour está dado por la suma de las longitudes $J_{c_i c_j}$ de todos los arcos de los cuales dicho tour está compuesto. Las distancias no necesariamente son simétricas (en un TSP asimétrico, $J_{c_i c_j}$ puede ser diferente de $J_{c_j c_i}$) ni el grafo está totalmente conectado. Si este es el caso, los arcos faltantes pueden ser añadidos asignándoles un costo infinito.

A continuación se presenta uno de los principales representantes de un algoritmos ACO, un Sistema de Hormigas (SH). Sin embargo, existe una variedad de algoritmos ACO que son extensiones del SH propuesto por [39] que serán descriptos al final de la presente sección.

EL SH fue el primer (1991) algoritmo ACO [47], el cual fue diseñado como un conjunto de tres algoritmos que diferían en la forma en que el rastro de feromona es modificado por las hormigas. Sus nombres son: *ant-density*, *ant-quantity*, y *ant-cycle*. Un número importante de algoritmos surgieron, inspirados por el *ant-cycle*, que fue el que mejor comporta-

miento mostró a través de diversas aplicaciones. Dado el éxito de este algoritmo, en la mayoría de los trabajos subsecuentes, se identifica al SH como un algoritmo de tipo *ant-cycle* por definición. La figura 3.3 presenta un esquema general de un SH para TSP, aunque como veremos más adelante, dicha estructura es lo suficientemente general que nos permitirá explicar variantes de los algoritmos ACO. El algoritmo

```

1. S_H()
2. {
3.   inicializar
4.   while (criterio_de_terminacion_no_sea_satisfecha)
5.     for  $k = 1$  to  $a$  do
6.        $j = \text{obtener\_ciudad\_inicial}()$ ;  $s^k = \langle j \rangle$ 
7.        $\mathcal{M}^k = \{j\}$ 
8.       while ( $|s^k| \neq n$ )
9.         selecciona ciudad  $j$  para ser incorporada
           con probabilidad  $P_{ij}^k = \frac{a_{ij}}{\sum_{h \in \mathcal{N}_i^k} a_{ih}}$ 
10.         $s^k = s^k \oplus \langle j \rangle$ 
11.         $\mathcal{M}^k = \mathcal{M}^k \cup \{j\}$ 
12.      end
13.      calcular  $J_\psi$  el costo de la solución generada  $\psi^k$ 
14.      guardar la mejor solución hasta el momento
15.    end
16.    modificar los niveles de rastro  $\tau_{ij}$  en todos los pasos
17.  end
18. }
```

Figura 3.3: El SH para TSP. Este algoritmo pertenece a la clase de algoritmos ACO.

puede ser descripto informalmente de la siguiente manera: Después de un proceso de inicialización (línea 2) éste itera por un número de clicos determinados por *criterio_de_terminacion_no_sea_satisfecha*. Un número m de hormigas es distribuida (podría ser en paralelo) sobre las n ciudades. El nodo de comienzo, o bien la ciudad inicial, puede ser elegido en forma aleatoria. \mathcal{M}^k , la memoria asociada a la hormiga k , es inicializada

añadiendo la ciudad actual al conjunto de ciudades ya visitadas (dicho conjunto es vacío inicialmente). Las hormigas entran en un ciclo (línea 8 \rightarrow 12) que dura n iteraciones, es decir, hasta que cada hormiga ha completado un tour.

Durante cada paso, una hormiga ubicada sobre el nodo i selecciona la próxima ciudad (línea 9) aplicando una regla de decisión que toma en cuenta \mathcal{N}_i^k . La solución en proceso de construcción y \mathcal{M}^k son modificadas en forma acorde.

Una vez que las hormigas han completado un tour (línea 16), pueden usar su memoria para evaluar la solución construida⁹ y recorrer en forma inversa el tour e incrementar la intensidad del rastro de feromona en los pasos que correspondan, es decir de las conexiones visitadas. Este proceso de refuerzo tiene el efecto de favorecer ciertas conexiones en futuros ciclos del algoritmo. Es importante destacar que el proceso de evaporación de la feromona es disparado una vez que todas las hormigas han terminado un tour.

Consideraciones acerca de las componentes de un SH

La cantidad de rastro de feromona τ_{ij} sobre la conexión l_{ij} tiene el fin de representar la conveniencia aprendida de ir a la ciudad j cuando se está en la ciudad i (lo que corresponde a la conveniencia que el arco pertenezca al tour construido por una hormiga). La información del rastro de feromona es cambiado durante el proceso de resolución del problema. Las hormigas depositan una cantidad de feromona que es proporcional a la calidad de las soluciones que generan. De tal forma, mientras más corto sea el tour generado por una hormiga, mayor será la cantidad de feromona que se deposita en los arcos usados para construir el tour. Este es uno de los elementos principales a través de los cuales el algoritmo dirige la búsqueda hacia regiones más promisorias del espacio de búsqueda.

La memoria (o estado interno) \mathcal{M}^k de cada hormiga contiene las ciu-

⁹Aunque en muchos casos, incluido TSP, la solución puede ser evaluada a medida que se va construyendo.

dades ya visitadas y es llamada la *lista tabu*¹⁰. La memoria \mathcal{M}^k es usada para definir, para cada hormiga k situada en la ciudad i , el conjunto de ciudades que ésta puede visitar desde allí, es decir el conjunto de ciudades factibles a visitar. A través de la consulta a su memoria interna \mathcal{M}^k , cada hormiga k puede construir soluciones factibles evitando visitar la misma ciudad dos veces. De la misma manera, la memoria puede ser usada por cada hormiga para computar el valor objetivo de la solución y la modificación del rastro de feromona, como fue explicado previamente.

La tabla de ruteo $\mathcal{A}_i = [a_{ij}(t)]$ del nodo i es obtenido siguiendo una composición funcional de los valores de rastro de feromona τ_{ij} y heurística $\eta_{ij} = 1/d_{ij}$, donde d_{ij} es la distancia entre las ciudades i y j . Esta es una de las principales heurísticas usadas en la resolución de TSP, aunque la misma puede variar, según el diseñador del algoritmo. Luego, cada componente de la tabla de ruteo es obtenido como $a_{ij} = \tau_{ij}(t)^\alpha \eta_{ij}^\beta$, donde α y β son dos parámetros que controlan el peso relativo del rastro de feromona y del valor heurístico. En términos generales, el rol de los parámetros α y β es el siguiente. Si $\alpha = 0$, las ciudades más cercanas serán seleccionadas con mayor probabilidad. De esta forma, el algoritmo SH se comportaría como un algoritmo greedy estocástico clásico (con múltiples puntos de comienzo, dado que las hormigas son distribuidas aleatoriamente sobre las ciudades). Por el contrario, si $\beta = 0$, solamente el aprendizaje a través del rastro de feromona es el que guía la búsqueda. Esta última situación llevaría a una convergencia prematura del algoritmo, situación en la cual todas las hormigas producen el mismo tour y que en general es subóptimo¹¹ [26]. Por lo tanto un balance apropiado entre la importancia dada a la intensidad del rastro de feromona y a los valores heurísticos debe ser considerado para un buen funcionamiento de este algoritmo.

Después que todas las hormigas han completado su tour, cada hormiga

¹⁰El término “lista tabú” es usado en este contexto para representar una memoria simple que contiene las ciudades ya visitadas y no existe relación alguna con la metaheurística Tabu Search.

¹¹Esto según los resultados de los experimentos aplicados a TSP.

k deposita un cantidad de feromona $\Delta\tau_{ij}^k(t) = Q/\mathcal{J}_\psi^k$ sobre cada conexión l_{ij} usada, donde \mathcal{J}_ψ^k es la longitud del tour $\psi^k(t)$ realizado por la hormiga k en la iteración t . El parámetro Q , una constante, es un parámetro del algoritmo.

$$\tau_{ij}(t) = \tau_{ij}(t) + \Delta\tau_{ij}(t), \quad \forall l_{ij} \in \psi^k(t), \quad k = 1, \dots, m \quad (3.1)$$

donde $\Delta\tau_{ij}(t) = \sum_{k=1}^m \Delta\tau_{ij}^k(t)$ y m representa el número de hormigas en cada iteración (en general se mantiene constante). El número total de hormigas es en general igual al número de componentes del problema, es decir $m = n^{12}$. La manera de calcular el valor $\Delta\tau^k(t)$ es una medición de la performance de la hormiga, dado que mientras más corto sea el tour encontrado, mayor será la cantidad de feromona depositada.

Luego que el proceso de actualización de feromona ha sido realizado, la evaporación de la feromona es llevado a cabo a través de la aplicación de la siguiente ecuación a todos los arcos del grafo G :

$$\tau_{ij}(t+1) = (1 - \rho)\tau(t) \quad (3.2)$$

donde $\rho \in (0, 1]$ es el coeficiente de evaporación del rastro de feromona ($1 - \rho$ es el coeficiente de persistencia). Es importante destacar que existen distintos métodos para realizar la actualización y evaporación de la feromona y muchos de ellos, son usados en las distintas versiones de los algoritmos ACO. Otro punto a destacar es que la cantidad inicial de feromona depositado en cada arco es representado generalmente por un valor constante positivo τ_o para todos los arcos. Sin embargo esto no es una regla, como lo demuestran las diversas aplicaciones de algoritmos ACO.

El Sistema de Hormigas y sus extensiones

A pesar de su buena performance, los SHs no eran competitivos con los algoritmos (estado-del-arte) para TSP. A fin de mejorar su performance,

¹²Aunque ésta es una cantidad sugerida y no es estricta.

la comunidad de investigación en este tópico comenzó a extender los SHs generando distintas versiones.

Una primera mejora, llamada *estrategia elitista*, fue introducida por Dorigo et al. en [48]. Dicha mejora consiste en dar al mejor tour desde el comienzo de la ejecución del algoritmo, un peso extra durante el proceso de modificación del rastro. El mejor tour se denota como T^{mtg} , donde *mtg* significa *mejor tour global*. La forma de llevar a cabo la estrategia elitista es la siguiente: cada vez que los rastros de feromona son modificados en la manera usual, se tiene en cuenta además aquellos arcos o conexiones que pertenecen al mejor tour global y se les añade una cantidad adicional de feromona. De esta manera, la ecuación 3.1 se transforma en:

$$\tau_{ij}(t) = \tau_{ij}(t) + \Delta\tau_{ij}(t) + \Delta\tau_{ij}^{mtg} \quad \forall l_{ij} \in \psi^k(t), \quad k = 1, \dots, m \quad (3.3)$$

donde,

$$\Delta\tau_{ij}^{mtg} = \begin{cases} e/L^{mtg} & \text{si el arco } (i, j) \in T^{mtg} \\ 0 & \text{en otro caso} \end{cases} \quad (3.4)$$

De esta manera, los arcos de tour T^{mtg} son reforzados con una cantidad de $e \cdot 1/L^{mtg}$, donde L^{mtg} es la longitud del tour T^{mtg} y e es un entero positivo. Es importante destacar que este tipo de modificación de la feromona es un ejemplo particular de las denominadas “acciones auxiliares” según fueron descritas en la sección 3.1.

Otros algoritmos producidos como resultado de las mejoras introducidas al SH original son, SH_{rank} , el Sistema de Colonia de Hormigas (SCH) y Sistema de Hormigas $\mathcal{MAX-MIN}$ (o $SH-\mathcal{MM}$). El algoritmo SH_{rank} [30] es de alguna forma una extensión de la estrategia elitista descrita anteriormente. El proceso que sigue es el siguiente: ordena las hormigas de acuerdo a la longitud de los tours que ellas generaron, y luego de cada fase de construcción de un tour, solamente las mejores $(w - 1)$ hormigas y la mejor global son las que aportan en el depósito

de feromona sobre los lados respectivos. Así, la r -ésima hormiga de la colonia contribuye a la actualización de la feromona con un peso dado por $\max\{0, w - r\}$, mientras que el mejor tour global refuerza los pasos con un rastro de feromona proporcional a un peso w . De esta manera la ecuación 3.3 se transforma en:

$$\tau_{ij}(t+1) = (1 - \rho) \cdot \tau_{ij}(t) + \sum_{r=1}^{w-1} (w - r) \cdot \Delta\tau_{ij}^r(t) + w \cdot \Delta\tau_{ij}^{mtg} \quad (3.5)$$

donde $\Delta\tau_{ij}^r(t) = 1/L^r$ y $\Delta\tau_{ij}^{mtg}(t) = 1/L^{mtg}$.

Por último, aunque no menos importante, se presenta el algoritmo SCH [46, 45] el cual surge a partir de Ant-Q [44]. La versión Ant-Q de la metaheurística ACO es un algoritmo desarrollado con el propósito de crear una conexión entre Aprendizaje por Refuerzo y Optimización Basada en Colonia de Hormigas. A partir de la experiencia en este tema, se mostró que algunos aspectos de Ant-Q, en particular lo referido a la regla de modificación de la feromona podían ser sobre simplificados sin afectar el desempeño del algoritmo. De esta manera, estudios relativos a Ant-Q fueron dejados en favor de uno más sencillo que incluía similarmente aspectos de Aprendizaje por Refuerzo. A diferencia de un SH, el algoritmo SCH incrementa la importancia de la explotación de la información recogida previamente por las hormigas con respecto al espacio de búsqueda y es llevado a cabo a través de dos principales mecanismos, la regla de transición y de actualización del rastro de feromona. La regla de transición hace referencia a la manera en que la próxima componente del problema será añadida a la secuencia en construcción¹³. Dicha regla es modificada para permitir explícitamente la exploración. Así una hormiga k que se encuentra en la ciudad i elige la ciudad j como la próxima ciudad a visitar siguiendo la siguiente regla:

¹³Recuerde el lector que esta explicación está vinculada a problemas cuya solución puede ser representada como una secuencia de componentes del problema. En el caso de TSP la secuencia es una permutación y es el problema que usamos para ejemplificar.

$$j = \begin{cases} \arg \max_{h \in \mathcal{N}_i^k} \{[\tau_{ih}(t)] \cdot [\eta_{ih}]^\beta\} & \text{si } q \leq q_0; \\ \hat{j} & \text{si } q > q_0 \end{cases} \quad (3.6)$$

donde q es una variable aleatoria uniformemente distribuida en $[0, 1]$, q_0 es un parámetro del algoritmo ($0 \leq q_0 \leq 1$) y $\hat{j} \in \mathcal{N}_i$ es una ciudad elegida en forma aleatoria de acuerdo a la siguiente probabilidad:

$$p_{i\hat{j}}^k = \frac{[\tau_{i\hat{j}}] \cdot [\eta_{i\hat{j}}]^\beta}{\sum_{h \in \mathcal{N}_i} [\tau_{ih}] \cdot [\eta_{ih}]^\beta} \quad (3.7)$$

la cual es muy similar a la usada en el SH. En definitiva, la regla de transición es idéntica al SH cuando $q \leq q_0$, pero es diferente cuando $q > q_0$. Este último caso está relacionado con la explotación del conocimiento disponible acerca el problema (es una regla greedy en relación a los valores del rastro y la heurística del problema). En primer caso, el algoritmo favorece la exploración, pero siempre guiado por la misma información. El parámetro es, en consecuencia, el que permitirá controlar el grado de exploración versus la explotación deseada. Una posibilidad es darle un valor y mantenerlo fijo a través de la ejecución. Otra alternativa, quizás más interesante, es hacer variar los valores de q desde 0 a 1 en forma progresiva de manera tal que se favorezca la exploración en las primeras iteraciones y la explotación en iteraciones más avanzadas [26].

La otra componente novedosa del algoritmo SCH es la regla de actualización del rastro. Basta recordar que en el SH original, todas las hormigas depositan rastros de feromona después de completar el tour (o completar la solución). Sin embargo, en el SCH, la única hormiga que puede depositar, en forma global, rastro de feromona es aquella que generó la mejor solución encontrada hasta el momento desde el comienzo de la corrida. En este sentido, la exploración está sesgada hacia aquellos tours similares al mejor encontrado. Otra diferencia con SH es que la modificación del rastro de feromona sólo se hace sobre los arcos involucrados en el mejor tour mencionado previamente. Esta regla (llamada

regla de modificación global) es de la forma:

$$\tau_{ij}(t) = (1 - \rho) \cdot \tau_{ij}(t) + \rho \cdot \Delta\tau_{ij}(t) \quad (3.8)$$

donde los (i, j) son los lados que pertenecen al mejor tour hasta el momento, ρ es el parámetro que gobierna la evaporación del rastro y,

$$\Delta\tau_{ij}(t) = 1/L^{mtg} \quad (3.9)$$

Dado que la regla anterior lleva a cabo una actualización global del rastro de feromona, existe en el algoritmo una regla de actualización local en la que participa toda la colonia. Esta regla local trabaja de la siguiente manera: cuando una hormiga se encuentra construyendo un tour y pasa de una ciudad i a una ciudad j , la concentración de feromona en la conexión (i, j) es modificada según:

$$\tau_{ij}(t) = (1 - \rho) \cdot \tau_{ij}(t) + \rho \cdot \tau_0. \quad (3.10)$$

El valor τ_0 es el mismo usado como valor inicial asignado al comienzo de la ejecución.

La idea de esta regla es que cuando una hormiga transite por una conexión (o arco del grafo) haga disminuir levemente la concentración de feromona. Esto trae como resultado que aquellas hormigas que transitan por conexiones ya visitadas, indirectamente favorezcan a las conexiones menos transitadas. En consecuencia, el algoritmo SCH evitará converger a un único paso dado que el objetivo de la regla local es que la preferencia por ciertas conexiones, aprendida por el algoritmo, vaya cambiando dinámicamente.

Sin duda para éste y toda la familia de algoritmos enmarcados en la metaheurística ACO existen muchas posibilidades en cuanto al diseño de nuevas versiones de los algoritmos anteriormente descritos, como así también la posibilidad de diseñar nuevos algoritmos o enfoques cuyo principio de diseño esté regido por el comportamiento cooperativo de los agentes que lo componen.

3.1.4. Aplicaciones de algoritmos ACO

Hasta el momento se han reportado una gran cantidad de aplicaciones de este enfoque que incluyen problemas de optimización combinatoria y numéricos, con y sin restricciones y otros tipos de problemas que incluyen diferentes grados de dinamismo. Esta situación es aún más destacable si se tiene en cuenta el tiempo relativamente corto de la aparición de este enfoque.

El problema más estudiado en este ámbito ha sido por mucho tiempo el TSP. Esto no sólo se debe a su estructura, la cual facilita la aplicación de algoritmos ACO, sino a su formulación, la cual es simple y fácil de entender. Sin embargo, estas características no cambian su naturaleza de ser un problema inherentemente difícil. También es importante destacar que la mayoría de los algoritmos ACO fueron diseñados considerando una estructura de grafo para el problema a resolver, aunque en la etapa de verificación incluyó instancias del TSP. Sin embargo, no pasó mucho tiempo para que la comunidad de investigadores comenzara a hacer los cambios necesarios en los algoritmos originales logrando rápidamente ampliar el espectro de problemas que podrían ser resueltos con este enfoque. Estos problemas incluyen: Problema de Asignación Cuadrática o QAP, Ruteo y redes de telecomunicación, Coloreo de Grafos, Redes ópticas, scheduling, etc. Varias de estas aplicaciones también incluyen propuestas de modelos e implementaciones paralelas para los algoritmos ACO, siendo esta rama una de las menos desarrolladas desde el punto aplicativo de esta metaheurística.

Estudios recientes, los cuales son parte de esta tesis, han reportado resultados de la aplicación de algoritmos ACO para resolver problemas numéricos con restricciones y problemas de optimización combinatoria, también con restricciones, los cuales no presentan una estructura que pueda ser visualizada a través de un grafo como lo sugiere la formulación original.

Otras aplicaciones van más allá del comportamiento basado en la co-

municación indirecta (característica que identifica a los algoritmos descritos aquí). Recientes aplicaciones de enfoques que se basan en otros aspectos del comportamiento de las colonias de hormigas han dado lugar a algunos desarrollos de algoritmos competitivos en el área de Minería de Datos y Procesamiento de Imágenes, entre otras.

3.2. Acerca del Teorema NFL (No-Free Lunch)

Como parte final del presente capítulo, es importante destacar un resultado muy importante en cuanto al uso y efectividad de las metaheurísticas para resolver problemas de optimización. En este sentido, es interesante mencionar el artículo de Schwefel ([9], Cap.3) acerca del Teorema NFL (Wolpert et al. [176]) el cual establece que no puede existir ningún algoritmo para resolver todos los problemas (por ejemplo de optimización) que sea en promedio superior a cualquier otro competidor. En dicho artículo, Schwefel realiza un análisis de esta aseveración en relación a los Algoritmos Evolutivos en particular, pero las apreciaciones vertidas son absolutamente válidas cuando dicha relación es establecida en un marco más general como el de las metaheurísticas. En consecuencia, la pregunta de si alguna metaheurística en particular es superior o inferior a cualquier otro enfoque alternativo, incluidas otras metaheurísticas, es una pregunta sin sentido. Lo que sí podría afirmarse es que ciertas metaheurísticas se comportan mejor que otros métodos con respecto a la resolución de clases de problemas específicos y por ende, éstas se comportan peor para otras clases de problemas.

El Teorema NFL puede ser corroborado en el caso de una gran variedad de metaheurísticas cuando son comparadas contra métodos clásicos de optimización dado que los últimos son muy eficientes para resolver problemas lineales, cuadráticos, fuertemente convexos, unimodales, separables y muchos otros. Sin embargo, las metaheurísticas en general pueden tener un muy buen comportamiento cuando las funciones (que

representan el problema a optimizar) son discontinuas, no diferenciables, multimodales, ruidosas, etc. La característica principal de muchas metaheurísticas es que su efectividad (o robustez) se extiende a un amplio campo de aplicaciones, aunque con una importantes pérdida de eficiencia cuando es aplicado a problemas simples para los cuales existen procedimientos específicos para su resolución.

En relación a la historia de los métodos para resolver problemas de optimización, es importante destacar que en la década de los '60 Schwefel [160] publica un par de algoritmos para búsqueda de óptimos en el *Computer Journal* y a partir de aquí surge un patrón de desarrollo de nuevos métodos que obedece aproximadamente a los siguientes pasos: El autor A publica un método y demuestra que es aplicable a determinado problema usando un conjunto e instancias de dicho problema para mostrar su efectividad. Luego el autor B aparece con un contra-ejemplo a través del cual pone en evidencia la debilidad del método del autor A. Además el autor B presenta una nueva técnica modificada la cual tiene un comportamiento superior que la anterior sobre los nuevas instancias consideradas. Y de esta manera el “juego” podría continuar indefinidamente. Indudablemente la mejor manera de clarificar esta situación debería en principio venir de la mano de un desarrollo teórico adecuado en el cual se debería definir claramente el dominio de aplicabilidad de cada algoritmo presentando pruebas de convergencia y resultados eficientes. Sin embargo, al momento de querer probar la propiedades de convergencia de ciertos algoritmos, esto sólo se puede lograr realizando muchas simplificaciones en torno a las situaciones a las cuales son confrontados, aunque esto no significa que un adecuado desarrollo teórico no pueda ser alcanzado. En la actualidad, para la mayoría de los métodos numéricos, muchas de las respuestas acerca de la capacidad de las diferentes metaheurísticas son contestadas a través de un conjunto, por lo general limitado, de instancias del problema y que muchas veces incluyen situaciones del mundo real con características no consideradas y la

mayoría de las veces muy comunes de encontrarlas. Tampoco existe un único catálogo de instancias de un determinado problema que sirvan para evaluar en forma concisa viejos y nuevos algoritmos de optimización.

En consecuencia, habrá siempre una dicotomía entre eficiencia y aplicabilidad general, entre confiabilidad y esfuerzo para resolver un problema, particularmente para los algoritmos de búsqueda de soluciones óptimas.

En síntesis, no puede existir un método que resuelva todos los problemas efectivamente y eficientemente, dado que esos objetivos son contradictorios. Además, si ya existe un método específico y eficiente para resolver un problema particular, no tiene mucho sentido usar una metaheurística. Sin embargo, si se pretende desarrollar un método muy específico y eficiente para un problema dado, puede ser un gran desafío para un teórico, pero el tiempo que podría llegar a consumir esta tarea puede ser prohibitivo desde el punto de vista práctico para una aplicación en particular. En este sentido, una técnica heurística no especializada y robusta podría ser, y frecuentemente lo es, de gran utilidad para la obtención rápida de buenas soluciones aunque no necesariamente las mejores de todo el espacio de búsqueda del problema.

Una de las grandes ventajas de las metaheurísticas en general, es que las mismas presentan un marco metodológico que es fácil de entender y manejar, y que la mismo tiempo pueden ser usadas como un método de caja negra o que puede estar abierto (a los efectos de ganar eficiencia) a la modificación y/o incorporación de nuevas componentes de manera que puedan alcanzar mayor sofisticación, especialización e inclusive hibridización. También, muchas de las metaheurísticas existentes son particularmente aptas para entornos dinámicos donde las restricciones y objetivos del problema van cambiando a través del tiempo.

Capítulo 4

Metaheurísticas y manejo de restricciones

4.1. Introducción

Este capítulo plantea una discusión acerca de los *métodos estocásticos-iterativos*, los cuales representan la base del comportamiento de los AEs y AHs, dos tipos de algoritmos encuadrados en aquellos que basan su comportamiento en el procesamiento de una población. La forma en que estos algoritmos representan las posibles soluciones del problema y la manera en que exploran el espacio de búsqueda son analizados en este capítulo. Asimismo, el capítulo incluye una descripción detallada de los distintos enfoques para el manejo de restricciones que han sido desarrollados últimamente para problemas de optimización en el contexto de algoritmos evolutivos y basados en colonias de hormigas.

4.2. Acerca de las metaheurísticas

Los métodos de optimización son herramientas usadas en diversos ámbitos de la vida real para los cuales uno de los objetivos principales es resolver un problema de optimización. Hasta el momento existe un gran cantidad de métodos de optimización que van desde algunos muy específicos, es decir aplicables a un clase pequeña y particular de proble-

mas; a aquellos más generales que pueden ser aplicables a una cantidad considerable de problemas de optimización. En el extremo de este espectro se encuentran los métodos de búsqueda estocástico-iterativos que incluyen muchas de las técnicas de optimización conocidas, entre éstas, las metaheurísticas objeto de estudio en esta tesis (la figura 4.1 muestra la forma general de este tipo de algoritmos).

1. Generar y evaluar un conjunto S de soluciones candidatas.
2. Producir y evaluar un conjunto nuevo S' de soluciones candidatas a través de cambios aleatorios a los miembros seleccionados de S .
3. Reemplazar algunos de los miembros de S por algunos miembros recientemente generados en S' , y volver al paso 2 (a menos que la condición de terminación haya sido alcanzada).

Figura 4.1: Búsqueda iterativa estocástica generalizada

Los métodos de búsqueda estocástico-iterativos tienden a ser favorecidos por tres razones principales. Primero, tienden a ser fácilmente implementables dado que solamente unas pocas líneas de código son en general necesarias para implementarlos. Segundo, son muy generales, es decir que no existe un requerimiento *a priori* sobre el tipo de problemas de optimización a los cuales puedan ser aplicados. Por ejemplo, la función que representa el problema a optimizar (*función objetivo*) no necesita ser diferenciable. Tercero, estos métodos se han aplicado exitosamente para encontrar buenas soluciones de una manera rápida para problemas duros de optimización correspondientes a aplicaciones académicas y del mundo real. El éxito de su aplicación está determinada por dos aspectos fundamentales. Primero, por lo general se les aplica a diversos problemas de optimización realizando cambios menores en el código. Segundo, dichas técnicas son lo suficientemente flexibles como para permitir la hibridización de heurísticas especializadas para explotar el conocimiento acerca del problema a resolver, lo cual define un método metaheurístico.

Todas las versiones de los métodos de búsqueda estocástico-iterativos están basadas en el principio de *generar- \mathcal{E} -evaluar* (*generate- \mathcal{E} -test*), el que incluye *búsqueda aleatoria* en su versión más simple. Aunque las versiones clásicas y más exitosas de los métodos de búsqueda estocástica-iterativa siguen este esquema, las versiones más avanzadas usan formas más interesantes y fructíferas para guiar la búsqueda a través del espacio de soluciones, como por ejemplo, simulated annealing y tabu search. En estos métodos la clave es usar de alguna manera la información de soluciones encontradas previamente para generar nuevas soluciones candidatas. Por lo tanto, el generador de soluciones candidatas es una componente crítica en estos métodos de optimización. De esta manera, para el diseño de un buen generador de soluciones candidatas se debe tener en cuenta lo siguiente: ¿Cómo se puede mejorar la calidad de las soluciones a generar usando información de soluciones generadas previamente? En los métodos de optimización actuales y bien consolidados tres ideas que han probado ser exitosas, están siendo usadas en respuesta a la pregunta anterior (ver figura 4.2).

Los métodos de optimización bien establecidos tales como simulated annealing, tabu search y algoritmos evolutivos (algoritmos genéticos, estrategias evolutivas, programación evolutiva, y programación genética) usan al menos unas de las ideas descritas en la figura 4.2. Dichos métodos incluyen además una gran variedad de características subsidiarias para controlar el curso de la búsqueda y adaptar el comportamiento del método a medida que el proceso de búsqueda avanza. Estos métodos, así como también otros basados en búsqueda estocástica-iterativa caen dentro de dos clases bien definidas: búsqueda local y basada en población. La búsqueda local (también denominada hill-climbing) itera esencialmente sobre la Idea 1, es decir que continuamente produce soluciones candidatas a través de pequeñas variaciones de la solución corriente. Al mismo tiempo, hill-climbing encapsula de alguna manera una variación de la Idea 3: la solución que conforma la base para generar nuevas so-

1. Nuevas soluciones candidatas son producto de pequeñas variaciones de soluciones previamente generadas. Estas variaciones pueden ser logradas por operadores conocidos como de *vecindad* o *mutación* y también conocidos como operadores de *movimiento*.
2. Nuevas soluciones candidatas son generadas a través de la recombinación de una o más soluciones candidatas, por ejemplo, el operador de *crossover*.
3. Los “padres” desde los que se generan nuevas soluciones candidatas (de acuerdo a los dos primeros puntos) son seleccionados del conjunto previo de candidatos a través de una estrategia estocástica y competitiva la cual favorece a aquellas soluciones de mejor calidad para conformar la base de una nueva solución candidata.

Figura 4.2: Ideas que subyacen al proceso de generación de nuevas soluciones

luciones es siempre la mejor solución encontrada hasta el momento. Sin embargo, métodos de búsqueda local más sofisticados como SA o TS relajan esta condición de diferentes maneras con el objetivo de mejorar el proceso de escalamiento (hill-climbing).

Por otro lado, el enfoque ACO, como fue descrito en la sección 3.1, es un método que trabaja asemejando la noción de soluciones candidatas con la ruta seguida por los agentes (hormigas) entre dos lugares. Luego, la fortaleza del rastro afectará la ruta a seguir por otros agentes. Este comportamiento puede ser visualizado [39] como una combinación de las Ideas 1, 2 y 3. Esencialmente, las rutas seguidas con anterioridad por los agentes (soluciones previas) afectarán (a través de la fortaleza del rastro) las soluciones generadas en el futuro por otras hormigas. Sin embargo, tal como fue establecido por Corne et al. [39], tiene más sentido verlo como lo que es, una idea nueva y novedosa para optimización inspirada por un proceso observado en la naturaleza (el comportamiento de una colonia de hormigas).

El tema central de la presente tesis es la aplicación de AEs y AHs a problemas de optimización con restricciones. Dichas técnicas metaheurísticas están basadas respectivamente en la evolución de las especies y el comportamiento de las colonias de hormigas y comparten varias características. A continuación se describen sus principales componentes con respecto a la implementación (una versión básica) de algunas de las ideas consideradas previamente pero bajo un esquema de búsqueda iterativo-estocástico generalizado (Fig. 4.1). También es considerada una pequeña instancia de TSP usada a través de la descripción. La instancia involucra 4 ciudades (ver figura 4.3). Los números en los cuadrados representan la distancia asociada a los respectivos arcos no dirigidos.

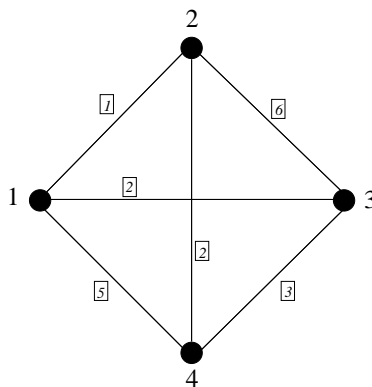


Figura 4.3: Instancia de TSP con 4 ciudades

Algoritmos Evolutivos

- Método iterativo: Cada iteración es llamada una “generación” en la cual son aplicados los operadores genéticos. El algoritmo puede ser diseñado con diferentes condiciones de parada o terminación, por ejemplo el número de generaciones [76, 114].

- Soluciones: Representan los individuos de la población y cada una de ellas es una potencial solución completa al problema a resolver.

Para nuestro ejemplo, las permutaciones $P_1 = (1\ 3\ 4\ 2)$, $P_2 = (4\ 1\ 3\ 2)$ y $P_3 = (3\ 1\ 4\ 2)$ representan soluciones válidas. Sin embargo, cuan-

do se usan decodificadores¹, ellos en sí mismos no son la solución, sin que codifican la forma en que la solución al problema dado es creada.

Un decodificador para TSP podría ser representado [114, 118] como un vector entero V de tamaño n donde $V[i] \in [1, n + i - 1]$. A fin de obtener la permutación respectiva, una lista $L = (1, 2, 3, 4)$ es tomada como referente; así, $V[i]$ indica que el $V[i]$ -ésimo elemento de la lista L debe ser añadido como próximo elemento en la permutación que se está construyendo. Por ejemplo, dado el individuo $V = (4, 1, 2, 1)$, la transformación para obtener la permutación procede en forma iterativa de la siguiente manera: al comienzo, toma el cuarto elemento de L como está indicado por $V[1]=4$. De esta manera la permutación en proceso de construcción es $P = (4)$ y $L = (1, 2, 3)$ (es decir que el cuarto elemento es eliminado de L). El proceso continúa barriendo los elementos restantes de V y por ende incorporando elementos a la permutación P de la siguiente manera:

$$\begin{array}{lll} V[2] = 1, & L = (1, 2, 3), & \text{y } P = (4 \ 1) \\ V[3] = 2, & L = (2, 3), & \text{y } P = (4 \ 1 \ 3) \\ V[4] = 1, & L = (3), & \text{y } P = (4 \ 1 \ 3 \ 2) \end{array}$$

Este procedimiento puede ser visto como un proceso paso-por-paso para construir una solución, de manera similar a lo realizado por un algoritmo ACO. De cualquier manera los decodificadores son los que conforman la población y por ende son recombinados y mutados de la misma manera que lo es cualquier solución representada de manera directa.

- **Enfoque competitivo, explotación:** Está basado en la supervivencia del más fuerte o adaptado a fin de guiar el curso de la búsqueda. En diferentes etapas del algoritmo, un proceso de selección es usado con

¹ *decodificadores* son un ejemplo de una representación indirecta de una solución y serán considerados en la sección 4.3.1 como una técnica alternativa de manejo de restricciones.

el objeto de decidir quién vive (para una posible reproducción) y quién es eliminado (cuando nuevos individuos, probablemente más adaptados, sean creados).

Por ejemplo, las tres permutaciones presentadas anteriormente tienen asociados los siguientes costos:

$$\text{cost}(P_1) = 8$$

$$\text{cost}(P_2) = 14$$

$$\text{cost}(P_3) = 15$$

Luego, el individuo P_1 tendrá más posibilidades de sobrevivir durante el proceso de selección para luego ser reproducido, y eventualmente, individuos como P_2 y P_3 tenderán a desaparecer de la población de soluciones candidatas.

- Exploración: Las nuevas soluciones son obtenidas básicamente a través de la aplicación de los operadores de crossover y mutación. En el caso de soluciones basadas en permutaciones, existen un gran cantidad de operadores que pueden ser aplicados para dar como resultado permutaciones (válidas). Por ejemplo, Partial Mapped Crossover (PMX), Order Based Crossover (OX), y Cycle Crossover (CX) son algunos ejemplos de operadores de crossover. Por otro lado, Swap, Inversión y k -opt son ejemplos típicos de operadores de mutación. Supongamos que se han implementado los operadores OX y Swap para el crossover y mutación respectivamente. OX crea un hijo tomando un subtour (previa generación de dos puntos de corte) de uno de los padres y preserva el orden relativo de las ciudades según aparecen en el otro padre. El otro hijo es generado en forma similar cambiando el rol de los padres, es decir, el que da la subsecuencia y el que fija el orden relativo a preservar. El operador Swap por su parte, genera dos posiciones diferentes en forma aleatoria e intercambia los respectivos valores.

Asumamos que las permutaciones P_2 y P_3 fueron seleccionadas para recombinación y los respectivos puntos de cruce generados son 3 y 4.

$$P_2 = (4\ 1\ |\ 3\ 2\ |)$$

$$P_3 = (3\ 1\ |\ 4\ 2\ |)$$

Después de la aplicación de OX son generados los siguientes hijos:

$$A = (1\ 4\ 3\ 2), \quad cost(A) = 15$$

$$B = (1\ 3\ 4\ 2), \quad cost(B) = 8$$

Claramente el hijo hereda información de la posición de uno de los padres y el orden relativo de las componentes desde el otro padre.

Supongamos que se aplica el operador de mutación al individuo A y las posiciones generadas aleatoriamente son 2 y 4. Luego, el hijo resultante es $A' = (1\ 2\ 3\ 4)$ con $costo(A') = 16$.

- Bloque de construcción: Soluciones de calidad incremental son obtenidas a través de la combinación de permutaciones parciales (subtour en TSP) de diferentes soluciones las cuales en promedio representan una adecuada solución al problema. De la misma manera, el operador de crossover, como el OX explicado anteriormente, se define de manera tal que el hijo resultante herede tanta información genética como sea posible de los padres involucrados (que pueden ser más de dos de acuerdo a los enfoques más recientes sobre aplicación de los operadores genéticos como *Multi-Crossover per Couple* [52, 53, 52, 55] y *Multi-Crossover Multiple-Parents* [56, 54, 40]). Luego, un hijo generado a través de OX hereda de uno de los padres, una copia exacta de una permutación parcial respetando las posiciones originales y, del otro padre, hereda las posiciones relativas de los elementos faltantes de la permutación. Por otro lado, el

operador de mutación puede ser visto como un operador que incorpora diversidad en la población cuando el algoritmo ha convergido a una región subóptima del espacio de búsqueda.

Algoritmos basados en Colonias de Hormigas

- Método Iterativo: Cada iteración es llamada un “ciclo”. De manera similar a los AEs, pueden ser usadas varias condiciones de parada. La más usual es “detener la búsqueda luego de un número máximo de ciclos”.
- Soluciones: En los algoritmos ACO, una solución no es un miembro de la población. Por el contrario, la población está representada por un conjunto de agentes u hormigas, es decir, los individuos que están a cargo de la construcción de las soluciones al problema.
- Enfoque cooperativo: Los agentes trabajan en forma independiente y al mismo tiempo, modifican la estructura global involucrada en el proceso de la construcción de las soluciones futuras (principio de *stigmergy* [39] o comunicación indirecta). La mejora en la calidad de las soluciones a través de los ciclos sucesivos influirá en la cantidad de feromona a ser depositada en las conexiones (arcos del grafo) que forman parte de aquellas soluciones mejoradas.

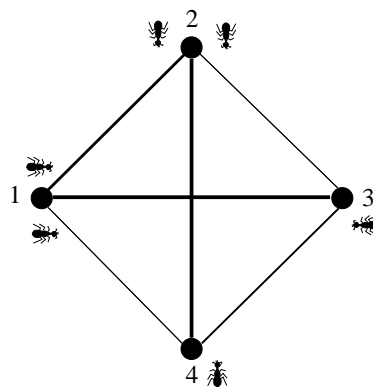


Figura 4.4: El grafo muestra la fortaleza del rastro de feromona sobre las conexiones.

La figura 4.4 representa un conjunto de seis agentes al comienzo del proceso de construcción durante un ciclo avanzado de la ejecución del algoritmo. Los seis agentes están distribuidos aleatoriamente en las ciudades involucradas. El espesor de los arcos representa la fortaleza del rastro de feromona sobre las conexiones respectivas en un ciclo determinado. En nuestro caso, las conexiones (1, 3), (2, 4), (1, 2), y (3, 4) son las más fuertes (es decir, un acumulación mayor de rastro de feromona).

- Exploración: Las soluciones son construidas paso-a-paso, añadiendo una componente del problema a la solución² basándose en información global (rastro) compartida por la colonia, y el conocimiento del problema (valor heurístico).

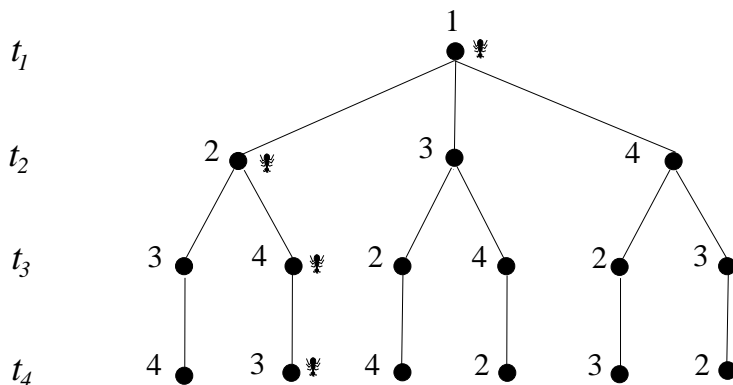


Figura 4.5: Espacio de búsqueda desde la perspectiva de un algoritmo ACO para una instancia de TSP donde la ciudad 1 es la inicial.

Es importante notar que el valor de la heurística y la información del rastro no son usados de una manera greedy³. Por el contrario, dichos valores son usados para construir una distribución de probabilidades para seleccionar la nueva componente a ser incorporada a la solución en proceso de construcción (ver línea 9 en la figura 3.3).

La figura 4.5 muestra el espacio de búsqueda para TSP visto desde

²Aunque esto no se mantiene para espacios continuos. Para más detalles ver sección 4.3.2.

³Sin embargo, existen variaciones como en un Sistema de Colonia de Hormigas (SCH) donde la elección greedy es una alternativa en el proceso de construcción de una solución.

la perspectiva del enfoque ACO (particularmente visto a través de un agente comenzando en la ciudad 1). En el tiempo t_1 la solución es representada por la permutación “parcial” (1). En el tiempo t_2 , existen tres ciudades posibles a las cuales visitar. Con una probabilidad alta, la ciudad 2 será elegida como la próxima dado que el rastro acumulado entre ellas es considerable y la distancia corta. En cada etapa del proceso de selección de la próxima ciudad, el agente realiza una “poda” del árbol de búsqueda para el ciclo actual. Este proceso es repetido hasta que se alcanza una hoja, es decir, hasta que la permutación sea completada.

- Bloque de construcción: La lógica detrás del enfoque ACO para problemas donde el orden de las componentes en la solución es importante, es el siguiente: la generación de mejores soluciones está influida por la cantidad de rastro depositado en las conexiones entre las componentes del problema (ver sección 3.1) además del valor heurístico asociado.

El rastro de cada conexión puede ser visto como una estructura generadora de soluciones (una matriz para TSP). Luego, mientras más reforzada esté una conexión entre dos componentes del problema, mayor será la probabilidad de que un agente construya una solución que contenga dicha componente.

4.3. Manejo de restricciones

Es claro que las restricciones son una parte integral de la formulación de cualquier problema de optimización. Dhar et al. [42] han escrito lo siguiente:

Casi todas las situaciones las cuales involucran una decisión, también involucran restricciones. Lo que distingue a los problemas de diferentes clases es la forma de las restricciones. Es decir, dependiendo de la forma en que el problema es visualizado, las restricciones pueden aparecer como reglas, dependencias de datos, expresiones algebraicas u otras formas.

Por ejemplo, la definición de NLP en la sección 1.1.1 es un ejemplo de un problema general de optimización para dominios continuos. Por otro lado, muchos problemas de optimización combinatoria también son restringidos. Por ejemplo, el problema de múltiples mochilas, problema de cobertura de conjunto, el conjunto independiente máximo, ruteo de vehículos, scheduling, etc.

En general, un espacio de búsqueda \mathcal{S} consiste de dos subconjuntos disjuntos de subespacios factibles y no factibles, \mathcal{F} y \mathcal{U} , respectivamente (figura 4.6). Esos subespacios no necesariamente son convexos y/o conectados entre sí. El proceso de resolución de un problema de optimización implica la búsqueda de una solución factible y óptima. Durante dicho proceso de búsqueda, se debe tener en cuenta la existencia de las soluciones factibles y no factibles que eventualmente puedan aparecer. Lamentablemente, el problema de tratar con soluciones factibles y no factibles independientemente del enfoque que está siendo usado, dista de ser trivial.

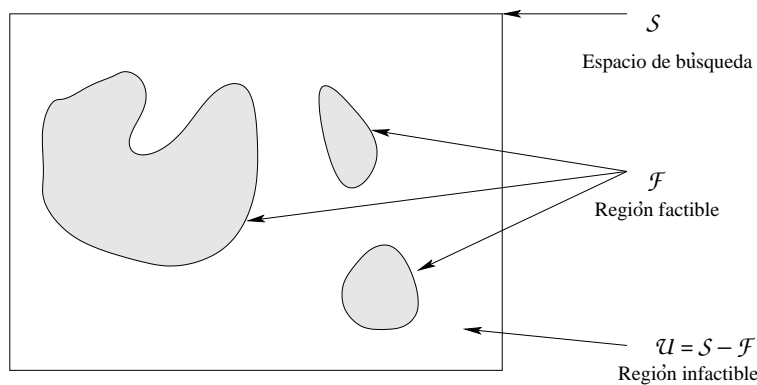


Figura 4.6: Un espacio de búsqueda hipotético con regiones factibles y no factibles.

En muchas técnicas metaheurísticas, la función de evaluación es la manera más directa de relacionar la calidad de las soluciones en la población actual con el problema a resolver. Por ejemplo, los mejores individuos dentro del enfoque evolutivo, son los que tendrán mayor probabilidad de sobrevivir y por ende, de ser elegidos para la reproducción o crossover. En el caso del enfoque ACO, la fortaleza del rastro se modifica según la calidad de las soluciones encontradas por los agentes en la colonia. Por lo tanto es un punto crucial que la función de evaluación esté bien definida de manera tal que capture y caracterice el problema de una manera razonable. Un desafío adicional puede presentarse cuando el problema a resolver incorpora restricciones ya que implica la muy probable aparición de soluciones no factibles durante el proceso de búsqueda. A pesar de que la solución final debería ser factible, podría ser útil permitir la existencia de soluciones no factibles mientras se busca la mejor solución factible. Luego, es indiscutible la importancia de la forma en que las soluciones factibles y no factibles son evaluadas y comparadas. Una falla de diseño en este proceso puede marcar la diferencia entre el éxito y el fracaso del método [118, 114].

La siguiente sección describe diferentes técnicas para el manejo de restricciones en relación al enfoque evolutivo y basado en colonia de hormigas.

4.3.1. Manejo de restricciones en AEs

Hay más de una pregunta a ser contestada en relación al espacio de búsqueda (figura 4.6) y la definición de la función de evaluación usada para medir o caracterizar individuos factibles y no factibles [118, 113]. Dichas preguntas incluyen las siguientes:

1. ¿Se deberían penalizar los individuos no factibles? En otras palabras, ¿se debería extender el dominio de la función objetivo f y asumir que la función de evaluación es $\text{eval}(n) = f(n) + \text{penalidad}(n)$?

Si es así, ¿cómo debería ser diseñada la función *penalidad*(*n*)?. En particular, ¿los individuos no factibles deberían considerarse inadecuados y por ende, eliminarlos de la población?

2. ¿Se debería cambiar la topología del espacio de búsqueda a través del uso de decodificadores?. Un decodificador interpreta y transforma al individuo en una solución factible.
3. ¿Se debería “reparar” una solución no factible con la idea de moverla (cambiarla) a una solución que se encuentre lo más cercana posible dentro del espacio factible?. En ese caso, ¿se debería reparar la versión reparada del individuo de la población (efecto Lamarckiano) o sólo usar un procedimiento de reparación con el propósito de evaluarla? (efecto Baldwiniano).
4. ¿Se debería comenzar con una población inicial de soluciones factibles y mantener la factibilidad de los hijos a través de la aplicación de operadores especializados?.
5. ¿Deberían manejarse por separado las soluciones y las restricciones?

En la actualidad, los métodos propuestos para el manejo de restricciones en algoritmos evolutivos son clasificados en cuatro grupos [118]. Aunque esta clasificación ha sido propuesta en relación a problemas de optimización de parámetros, algunos de los métodos son adecuados para su aplicación en la resolución de problemas de optimización combinatoria. Dichos métodos incluyen:

- Métodos basados en la preservación de la factibilidad de las soluciones.
- Métodos basados en decodificadores.
- Métodos basados en funciones de penalidad.

- Métodos que hacen una distinción bien clara entre soluciones factibles y no factibles.
- Métodos híbridos.

Métodos basados en la preservación de la factibilidad de las soluciones

Un ejemplo bien conocido de este enfoque es el sistema GENOCOP propuesto en [116, 117]. El diseño del sistema está basado en el uso de operadores especializados los cuales transforman individuos no factibles en individuos factibles. En términos matemáticos, es equivalente a decir que \mathcal{F} , la región factible, es cerrada bajo la aplicación de dichos operadores especializados. Este método asume restricciones lineales y solamente una solución factible inicial (o una población inicial factible). Las ecuaciones lineales son usadas para eliminar algunas variables, dado que éstas pueden ser reemplazadas por combinación lineal de las variables remanentes. Las desigualdades son modificadas de manera similar. Con el objeto de mantener la factibilidad de las soluciones, se define un conjunto de operadores que hacen cerrado al conjunto de soluciones factibles. Por ejemplo, cuando una componente particular x_i de un vector⁴ (o solución) \mathbf{x} es mutado, el sistema determina su dominio corriente $dom(x_i)$ y luego el nuevo valor para x_i es obtenido de este dominio usando alguna distribución de probabilidades. Por otro lado, el crossover aritmético (ver [114]) $a\mathbf{x} + (1 - a)\mathbf{y}$, aplicado a vectores (individuos) factibles \mathbf{x} e \mathbf{y} da siempre como resultado una solución factible (con $0 \leq a \leq 1$) en espacios de búsqueda convexos (dado que el sistema asume restricciones lineales solamente, esto implica convexidad del espacio de soluciones factibles).

Trabajos recientes en sistemas que realizan la búsqueda solamente en la frontera entre la zona factible y no factible constituyen un ejemplo adicional de este enfoque. La sección 5.2 provee un discusión detallada

⁴En GENOCOP cada individuo es representado por un vector real.

sobre este tópico el cual está directamente relacionado al objetivo de esta tesis.

Métodos basados en funciones de penalidad

Las funciones de penalidad han sido una parte importante de la literatura de problemas de optimización restringidos en las últimas décadas. Éste es uno de los enfoques más usados en el ámbito de la comunidad de computación evolutiva para atacar problemas restringidos, en donde se usa el concepto de penalidad externa, es decir, sólo las soluciones no factibles son penalizadas.

En general, el método de penalidad está basado en la distancia de una solución de la región factible, o bien, basado en el esfuerzo para reparar la solución, es decir, para forzarla a volver a la región factible \mathcal{F} . El primer caso es el más popular. En muchos métodos un conjunto de funciones f_i ($1 \leq j \leq m$) es usado para construir la penalidad, donde la función f_j mide la violación de la j -ésima restricción de la siguiente manera:

$$f_j(\mathbf{x}) = \begin{cases} \max\{0, g_j(\mathbf{x})\}, & \text{if } 1 \leq j \leq q \\ |h_j(\mathbf{x})| & \text{if } q + 1 \leq j \leq m. \end{cases}$$

Sin embargo, estos métodos difieren en muchos aspectos importantes, como por ejemplo, en cómo se diseña la función y cómo se aplica ésta a las soluciones no factibles. Es importante hacer notar que en optimización numérica existe un teorema según el cual si se incrementa sistemáticamente la penalización, se garantiza llegar a la zona factible. Es importante, entonces, hacer notar que las funciones de penalización pueden ser efectivas para conducirnos a la zona factible pero no necesariamente a la frontera con la región no factible que es el tema de esta tesis.

A continuación se describen brevemente cada uno de ellos:

- Penalidades estáticas.

Un método simple para penalizar soluciones no factibles es aplicar una penalidad a las soluciones que violan las restricciones de alguna manera. Existen muchas alternativas para implementar este enfoque considerando el número y tipo de restricciones. Por ejemplo, el método propuesto por Homaifar et al. [89] asume que para cada restricción se establece una familia de intervalos los cuales determinan un coeficiente apropiado de penalización. Este método considera un número k de niveles de violación para cada una de la m restricciones del problema. Por lo tanto, se establece un conjunto de coeficientes R_{ij} para $(i = 1, \dots, k; j = 1, \dots, m)$. Dicho método trabaja de la siguiente manera:

- para cada restricción, crear varios (k) niveles de violación,
- para cada nivel de violación y para cada restricción, crear un coeficiente de penalidad R_{ij} para $i = 1, 2, \dots, k$ y $j = 1, 2, \dots, m$; de manera tal que niveles más altos de violación requerirán valores más grandes para el coeficiente.
- comenzar con una población de individuos generada aleatoriamente (incluyendo factibles y no factibles)
- evolucionar la población, y evaluar los individuos usando la siguiente fórmula:

$$eval(\mathbf{x}) = f(\mathbf{x}) \sum_{j=1}^m R_{ij} f_j^2(\mathbf{x}).$$

A pesar de su simplicidad, la principal desventaja de este método es el número de parámetros a considerar. Por ejemplo, para m restricciones, el método requiere $m(2k + 1)$ parámetros. Un conjunto limitado de experimentos reportado por Michalewicz [113] indica que el método puede proveer buenos resultados si los niveles de las violaciones y los coeficientes de penalización R_{ij} son ajustados al problema bajo consideración. Otras alternativas para penalizar soluciones no factibles usan métricas basadas en la distancia de la

solución a la región factible del espacio de búsqueda. De hecho, muchos investigadores en computación evolutiva han explorado variaciones de funciones de penalidad estáticas basadas en la distancia (por ejemplo, Bačk et al. [12], Richardson et al. [148], Huang et al. [90], Goldberg [76]).

■ Penalidades dinámicas:

Una variación de la función de penalización basada en la distancia es incorporar un aspecto dinámico que incremente la severidad de la penalidad para una distancia dada, a medida que la búsqueda progresa. Este enfoque tiene la propiedad de permitir soluciones altamente factibles en los inicios de la ejecución del algoritmo, y a medida que avanza la búsqueda, se incrementa la penalización impuesta a las soluciones no factibles de manera tal que se guía la búsqueda a la región factible del espacio de búsqueda. En el método propuesto por Joines et al. [93], los individuos son evaluados (en la generación t) a través de la siguiente fórmula:

$$eval(\mathbf{x}) = f(\mathbf{x}) + (C \times t)^\alpha \sum_{j=1}^m f_j^\beta(\mathbf{x}),$$

donde C , α y β son constantes. Este método requiere un número mucho más pequeño de parámetros en comparación con el método descrito anteriormente (penalización estática). Además, en vez de definir varios niveles de violación, la presión sobre las soluciones no factibles es incrementada hacia el final del proceso de búsqueda; es decir, la componente $(C \times t)^\alpha$ alcanza valores más altos a medida que t se incrementa.

Es importante resaltar que este método provee, en algunos casos, penalidades drásticas: muchas veces el factor $(C \times t)^\alpha$ crece muy rápido para considerarlo útil. Según experimentos reportados por Michalewicz [113], se puede observar que el sistema tiene muy pocas

oportunidades de escapar de óptimos locales, por lo tanto el mejor individuo es encontrado en las primeras generaciones.

■ Penalidades “annealed”

Esta es una variación del método de penalización dinámico. Fue descrito bajo el nombre de GENOCOP II (Michalewicz, et al. [115] y Michalewicz [114]). Esta versión modificada trabaja según se detalla a continuación:

- dividir todas las restricciones en cuatro subconjuntos: ecuaciones lineales, desigualdades lineales, ecuaciones no lineales, y desigualdades no lineales,
- seleccionar un único punto (solución) en forma aleatoria (la población inicial consiste de copias del mismo individuo). Este punto inicial debe satisfacer las restricciones lineales,
- dar un valor inicial a la temperatura, $\tau = \tau_0$,
- evolucionar la población usando la siguiente fórmula:

$$eval(\mathbf{x}, \tau) = f(\mathbf{x}) + \frac{1}{2\tau} \sum_{j=1}^m f_j^2(\mathbf{x})$$

- si $\tau < \tau_f$, parar, en otro caso
 - decrementar la temperatura τ ,
 - la mejor solución sirve como punto de partida para la próxima generación,
 - repetir los pasos previos del algoritmo.

El método distingue entre restricciones lineales y no lineales. Además, mantiene la factibilidad de las soluciones sobre todas las restricciones lineales usando un conjunto de operadores genéticos cerrados usando operadores especializados. En cada iteración el algoritmo considera las restricciones activas solamente. La presión sobre las

soluciones no factibles es incrementada según disminuyen los valores de la temperatura τ . El método tiene la característica adicional de comenzar con un único punto; sin embargo, según Michalewicz [118], esta característica no es esencial para el algoritmo. Otro de los datos necesarios es la temperatura inicial y la final (de congelamiento) llamadas τ_0 y τ_f respectivamente, y también la consideración de un esquema de enfriamiento de dicha temperatura. Algunos valores estándar para estos parámetros han sido reportados por Michalewicz et al. [115] y son los siguientes: $\tau_0 = 1$, $\tau_{i+1} = 0,1 \cdot \tau_i$, y $\tau_f = 0,000001$.

■ Penalidades adaptativas:

Este método fue propuesto por Bean et al. [17]. Es similar al método que usa penalización “annealed”, sin embargo, en este caso una componente de la función de penalidad se retro-alimenta a partir del proceso de búsqueda. Así, cada individuo es evaluado a través de la siguiente fórmula:

$$eval(\mathbf{x}) = f(\mathbf{x}) + \lambda(t) \sum_{j=1}^m f_j^2(\mathbf{x}),$$

donde $\lambda(t)$, el valor de retro-alimentación, es modificado en cada generación t de la siguiente manera:

$$\lambda(t+1) = \begin{cases} (1/\beta_1) \cdot \lambda(t), & \text{si } \mathbf{b}^i \in \mathcal{F} \\ & \text{para todo } t-k+1 \leq i \leq t \\ \beta_2 \cdot \lambda(t), & \text{si } \mathbf{b}^i \in \mathcal{S} - \mathcal{F} \\ & \text{para todo } t-k+1 \leq i \leq t \\ \lambda(t), & \text{en otro caso} \end{cases}$$

donde \mathbf{b}^i denota el mejor individuo en la generación i , según la función $eval$. $\beta_1, \beta_2 > 1$ y $\beta_1 \neq \beta_2$ para evitar ciclos. La función de penalización adaptativa fue también usada por Smith et al. [163],

donde fueron incorporadas el avance en el proceso de búsqueda y la severidad en las restricciones como información. Hamida et al. [82] proponen ASCHEA⁵, un algoritmo adaptativo para problemas de optimización restringidos el cual incorpora una función de penalidad cuya adaptabilidad está basada en información obtenida de la población. Básicamente lo que el método propone es ajustar un coeficiente de penalidad durante el proceso de adaptación. En [83], ASCHEA es extendido de la siguiente manera: por un lado, se consideran múltiples coeficientes, uno para cada restricción, de manera tal que la adaptación de su valor es independiente; por otro lado, se agrega un método para conformar nichos a fin de manejar funciones multimodales. Otras aplicaciones del método incluyen algunos trabajos de Gen et al., como por ejemplo [69] en donde se aplica un AG para diseño de topologías de redes.

■ Penalización mortal.

Este es uno de los métodos más populares y fáciles de implementar en AEs (especialmente en las Estrategias Evolutivas [11]) el cual consiste en la eliminación de soluciones no factibles de la población. Este método trabaja bien si el espacio de búsqueda factible es convexo y constituye una parte razonable de todo el espacio de búsqueda. De otra manera, este enfoque tendría serias limitaciones debido a que el algoritmo evolutivo estaría la mayor parte del tiempo tratando de generar soluciones factibles con muy pocas oportunidades de explorar efectivamente y eficientemente el espacio de búsqueda.

Un método alternativo basado en funciones de penalización es el denominado algoritmo genético segregado propuesto por Le Riche et al. [149] como una manera diferente de manejar la robustez del nivel de penalidad: dos funciones de fitness diferentes con términos de penalidad estática p_1 y p_2 (uno mayor que el otro). La idea principal es que el enfoque re-

⁵Adaptive Segregational Constraint Handling Evolutionary Algorithm.

sultará en mantener dos subpoblaciones: los individuos elegidos según f_1 se situarán con mayor probabilidad en la región no factible, mientras que aquellos elegidos según f_2 lo harán en la región factible. De esta manera lo que se pretende es alcanzar el óptimo (en la región factible) aproximándolo desde ambos lados de la frontera del espacio factible y no factible.

Por su parte, Runarsson et al. [151], proponen un nuevo enfoque denominado Ranking Estocástico (*Stochastic Ranking*) el cual establece un balance estocástico entre la función objetivo y las funciones de penalidad. Adicionalmente presenta una nueva visión sobre los métodos basados en penalidades. Dicha visión relaciona en términos de dominancia los respectivos valores de la función objetivo y de penalidad. Más específicamente, el método apunta a eliminar los factores de penalización cuyos valores adecuados, son en general, difíciles de encontrar. El método puede ser encuadrado dentro del enfoque de penalización externa. Los resultados reportados en [151] muestran la efectividad y eficiencia de este método sobre un conjunto de 13 casos de prueba ampliamente conocidos y muchos de ellos objeto de estudio en la presente tesis. Otro trabajo de los mismos autores, propone además del ranking estocástico, un enfoque denominado Ranking Competitivo Global [152] en el cual el ordenamiento se realiza haciendo una comparación con todos los individuos de la población.

Métodos basados en decodificadores

Los decodificadores ofrecen una opción interesante a la comunidad de CE. En estas técnicas, un cromosoma “da las instrucciones” sobre cómo construir una solución factible. Por ejemplo, una secuencia de items para el problema de la mochila (knapsack) puede ser interpretado como: “tomar un item si es posible”, lo que llevaría siempre a una solución factible [114] de manera similar al ejemplo de TSP dado previamente en la sección 4.2.

Sin embargo, es importante destacar que varios factores deberían ser tomados en cuenta cuando se usan los decodificadores. Cada decodificador impone un mapeo $T : \mathcal{D} \rightarrow \mathcal{F}$ entre una solución factible del espacio de decodificadores \mathcal{D} y una solución decodificada correspondiente al espacio factible \mathcal{F} . Es importante que varias condiciones sean satisfechas: (1) para cada solución $s \in \mathcal{F}$ existe un decodificador d que la representa, es decir que T sea suryectivo; (2) cada solución codificada $d \in \mathcal{D}$ tiene una solución factible $s \in \mathcal{F}$, es decir, T es total; y (3) todas las soluciones en \mathcal{F} deberían ser representadas por el mismo número de codificaciones d , es decir que el grado de redundancia debería ser idéntico para todas. Lo ideal es que no exista redundancia, es decir que T sea un mapeo biyectivo. Además de las tres condiciones previas, puede ser útil considerar otro tipo de situaciones considerando los estudios de Palmer et al. [125]: es razonable requerir que (4) la transformación T sea eficientemente computada (computacionalmente rápida) y (5) que posea características de localidad en el sentido que pequeños cambios en la solución codificada resultarán en pequeños cambios en la solución en sí misma (es decir, las distancias entre las soluciones deberían ser comparables en cada uno de los espacios considerados).

En el caso de los espacios continuos, los decodificadores no han sido investigados en profundidad. Como un ejemplo, es posible mencionar el enfoque de Koziel et al. [103, 104] para resolver problemas de optimización restringidos. Este enfoque incorpora un mapeo homomórfico entre un cubo n -dimensional y un espacio factible. El mapeo transforma el problema restringido en uno no restringido. Este método tiene muchas ventajas en relación a los métodos descritos anteriormente: no se necesitan parámetros adicionales, no hay necesidad de evaluar o penalizar soluciones no factibles, dado que no existen; y una facilidad adicional, la de poder aproximar una solución en el borde de la región factible sin necesidad de operadores especializados. Sin embargo, encontrar un mapeo adecuado puede ser una tarea ardua y altamente dependiente del

problema. Un ejemplo interesante de uso de decodificadores en el ámbito de optimización combinatoria es el artículo de Gottlieb et al. [79] el cual presenta un AE basado en decodificadores para resolver MKP. El estudio está orientado a mostrar la importancia de conseguir una medida de la “localidad”⁶ del decodificador usado para lograr soluciones de alta calidad. Otros aspectos considerados son la influencia en la solución encontrada según el sesgo inducido por la heurística elegida y por último, la importancia de mantener una adecuada diversidad en la población. También es importante destacar que este estudio de Gottlieb et al. [79] es posterior a otros estudios del mismo autor en donde enfatiza la importancia de la búsqueda en la frontera. Detalles de estos trabajos serán dados en la sección 5.3.

Métodos basados en la búsqueda de soluciones factibles

Existen métodos que enfatizan la distinción entre soluciones factibles y no factibles en el espacio de búsqueda \mathcal{S} . En esta sección se describe tres de ellos:

- Método de memoria de comportamiento.

El método, propuesto por Schoenauer et al. [158], considera las restricciones del problema en secuencia; el cambio desde una restricción a otra es hecha en función del arribo de un número suficiente de individuos factibles en la población.

- Método basado en la superioridad de los puntos factibles.

Este método, desarrollado por Powell et al. [134] está basado en el enfoque de penalización clásico, pero con una notable excepción. Cada individuo es evaluado por la fórmula:

$$eval(\mathbf{x}) = f(\mathbf{x}) + r \sum_{j=1}^m + \theta(t, \mathbf{x}),$$

⁶Pequeños cambios en el genotipo deberían reflejar pequeños cambios en el fenotipo.

donde r es una constante; sin embargo, la componente original $\theta(t, \mathbf{x})$ es una función adicional dependiente de la iteración la cual influye la evaluación de las soluciones no factibles. Luego, el método distingue entre soluciones factibles y no factibles a través de una regla heurística propuesta por Richardson et al. [148]: para cualquier individuo factible \mathbf{x} y para cualquier individuo no factible \mathbf{y} : $eval(\mathbf{x}) < eval(\mathbf{y})$; esto es, cualquier solución factible es mejor que cualquier solución no factible⁷. Por otro lado, K. Deb [41] ha propuesto una modificación a este enfoque. La función de evaluación es cambiada de la siguiente manera:

$$eval(\mathbf{x}) = \begin{cases} f(\mathbf{x}), & \text{if } \mathbf{x} \text{ es factible,} \\ f_{max} + \sum_{j=1}^m f_j(\mathbf{x}) & \text{en otro caso,} \end{cases}$$

donde f_{max} es el valor de la función de la peor solución factible en la población. Es importante remarcar que para las soluciones no factibles, la función objetivo no es considerada. Este enfoque, para problemas de diseño en ingeniería y otros, ha mostrado ser mejor que el método propuesto por Powell et al. [134] para encontrar soluciones óptimas.

■ Reparación de individuos no factibles

Un ejemplo de este método es el sistema Genocop III propuesto por Michalewicz et al. [119]. Genocop III incorpora el sistema Genocop original pero extendiéndolo a través del manejo de dos poblaciones separadas, donde la evolución en una población influye la evaluación de los individuos en la otra población. La primera población P_s consiste de los denominados “puntos de búsqueda” de S los cuales satisfacen las restricciones lineales del problema. La segunda población P_r consiste de los puntos de referencia del conjunto \mathcal{F} ; dichos puntos son factibles. Los puntos de referencia \mathbf{r} de P_r son

⁷Para problemas de minimización.

evaluados directamente a través de la función objetivo, y los puntos de búsqueda de P_s son “reparados” para la evaluación usando un método iterativo el cual involucra un punto de referencia. Otro ejemplo muy interesante con respecto a la aplicación de un proceso es reparación, son los trabajos de Chu et al. [33, 18, 32, 19]. En este caso, el cromosoma es una cadena de bits que representa la solución a problemas discretos restringidos (Múltiples Mochilas, Cobertura de Conjuntos y Cobertura de Nodos) algunos de ellos objeto de estudio de la presente tesis. Los resultados mostrados en dichas aplicaciones muestran la efectividad de un AG cuando éste incorpora un proceso de reparación usando heurísticas asociadas a cada uno de los problemas considerados. En el trabajo de Raidl [138] se presenta un AG híbrido para resolver MKP. Por un lado a la población inicial se le aplica un proceso de pre-optimización basado en una solución del problema relajado (usando Programación Lineal). Adicionalmente el GA incluye operadores avanzados de búsqueda local y reparación, lo cuales son aplicados a cada nueva solución generada. En este trabajo de compara principalmente la disminución lograda por el algoritmo propuesto en comparación con uno de los mejores AGs para MKP propuesto por Chu [32]. Lo mas importante a destacar, más allá de la reparación utilizada, es que la aceleración en la convergencia lograda está dada principalmente por el mecanismo de inicialización de la población, esto es, la población inicial contiene un conjunto de soluciones de alta calidad obtenidas a partir de una solución del problema relajado.

Métodos Híbridos

En esta clase se pueden mencionar algunos enfoques que combinan técnicas de computación evolutiva con procedimientos determinísticos para problemas de optimización numérica. Por ejemplo, Waagen et al. [170] combinan un AE con el método del conjunto directriz de Hooke-Jeeves

probado con tres funciones no restringidas. Myung et al. [123] presentaron un enfoque similar pero estudiando problemas restringidos. Otros métodos híbridos (ver Parmee et al. [131] y Schaffer [155]) usan los valores de la función objetivo f y las penalizaciones f_j ($j = 1, \dots, m$) como elementos de un vector y se aplican técnicas de optimización multiobjetivo para minimizar todas las componentes del vector. Surry et al. [166] han propuesto un enfoque en donde todos los miembros de la población son ordenados sobre la base del grado de violación a las restricciones. Dicho ordenamiento r , junto con el valor de la función objetivo f , se transforma en un problema de optimización de dos objetivos.

Otros métodos híbridos interesantes incluyen, entre otros, algoritmos culturales (Reynolds [146] y Reynolds et al. [147]). Este tipo de algoritmos están basados en la combinación de un método de búsqueda (por ejemplo, un algoritmo evolutivo) y un esquema de representación de conocimiento de manera tal de poder capturar la experiencia individual dentro de un espacio de creencia para dirigir la búsqueda. Coello et al., [31, 37] proponen un enfoque basado en algoritmos culturales en el cual el conocimiento a obtener está constituido por un mapa de la región factible a fin de dirigir la búsqueda. Un enfoque similar propuesto por Ray et al. [139] basa su diseño en la interacción social de los individuos, sea ésta intra-social o inter-social, lo que deriva en las siguientes observaciones de gran importancia: *Una civilización emerge y avanza gracias a las relaciones (sic) cooperativas entre las sociedades que la componen y además, los individuos en cada sociedad interactúan unos con otros con el objetivo de mejorar y/o progresar.*

Similarmente, a través de un mecanismo de coincidencia de patrones (pattern matching) [87], se pretende utilizar el conocimiento sobre cuáles son aquellas restricciones que son violadas y cuáles no. Este conocimiento es incorporado en la función de fitness de manera tal de reflejar la calidad de una solución según el número de restricciones no satisfechas, el grado de violación de las mismas y el valor objetivo en sí mismo. A partir de

estos valores (de los cuales se puede deducir la factibilidad y calidad de los individuos) se realiza un proceso de selección específico para aplicar los operadores genéticos. El AG usado aquí se denominó sistema CONGA y fue comparado con GENOCOP II y III para cinco problemas $G1$, $G7$, $G9$, $G10$ y $G13$ [118].

Dentro de los enfoques híbridos también se puede mencionar a GA-MGA (Genetic ALgorithm & Micro-Genetic Algorithm) [95]. Este enfoque combina un AG estándar con un Micro-AG, el cual se podría definir como un algoritmo genético con una población pequeña sobre la cual se realiza un breve evolución. La idea principal de este enfoque es que el AG principal realiza la búsqueda global, mientras que el Micro-AG es el encargado de explorar el vecindario de las soluciones que van siendo encontradas por el primero. El Micro-AG es similar a un *hill-climber*; sin embargo, la búsqueda que realiza no es a través de pasos independientes en cada dimensión del espacio de búsqueda, sino que realiza una “búsqueda local genética”. La principal característica del Micro-AG es la de poder seguir un paso o camino apropiado hacia el óptimo sobre funciones que exhiben un “paisaje” (landscape) complejo. GA-MGA (su nombre original) fue comparado con doce métodos distintos ampliamente conocidos y usados para manejo de restricciones en algoritmos evolutivos. La calidad de los resultados de GA-MGA muestra un desempeño muy satisfactorio no sólo considerando los mejores valores encontrados, sino también desde el punto de vista de los valores promedios (media y mediana), peores y porcentaje de soluciones factibles encontradas en cada ejecución.

Con respecto al problema de satisfacción de restricciones (CSP⁸), Paredis [130] propone un enfoque en el cual dos poblaciones de individuos que representan respectivamente restricciones y soluciones potenciales al problema, interactúan de una manera similar a la que lo hacen un depredador y presa en la naturaleza. El método se denomina CCS (Co-evolution

⁸Siglas en Inglés de *Constraint Satisfaction Problem*.

approach to Constraint Satisfaction) el cual combina el método LTFE (o Life-Time Fitness Evaluation) y co-evolución. LTFE es una manera de evaluar a un individuo a través de una serie de verificaciones durante su tiempo de vida. En este caso, se aproxima el fitness de cada individuo (cualquiera sea la población a la que pertenezca) como la suma de los “pagos” recibidos durante las últimas 25 generaciones. El pago recibido en cada generación es proporcional al grado de factibilidad del individuo.

El enfoque planteado por Tan et al. [168] para manejo de restricciones, realiza una transformación angular que permite manejar las restricciones no lineales a nivel de genes o codificación. De esta manera se intenta enfocar la búsqueda sólo en el espacio factible dado que la representación angular por sí sola daría lugar a individuos factibles sin necesidad de operadores de crossover o mutación especializados. Este método fue aplicado sobre dos problemas, el conocido *G3* descrito en [104] y uno especialmente diseñado por los autores del método.

Otra propuesta alternativa para manejo de restricciones en algoritmos evolutivos consiste en considerar a las restricciones como objetivos, es decir, un problema mono-objetivo se transforma en uno multiobjetivo sin restricciones. Montes et al. [112] presentan un estudio comparativo de cuatro técnicas propuestas con anterioridad. Dichas técnicas son: CO-MOGA [165], el uso de VEGA⁹ según Coello [36], NPGA (Niche-Pareto Genetic ALgorithm) [34] y por último, la que utiliza MOGA [35]. Los resultados encontrados vistos desde una perspectiva global para todos los problemas considerados, muestran un comportamiento satisfactorio de las mismas, aunque en muchos de los casos no fueron capaces de alcanzar los valores óptimos. Los autores concluyen que “... los conceptos de optimización evolutiva multiobjetivo pueden resolver problemas con un solo objetivo en espacios restringidos. Sin embargo, se requiere de un mecanismo adicional para mejorar el desempeño de estos enfoques, ya que tiene obvios problemas para alcanzar el óptimo global en varias de las

⁹Vector Evaluated Genetic Algorithm propuesto por Schaffer en [156].

funciones de prueba utilizadas ...”. En esta dirección, Hart et al. [57] proponen un técnica para manejo de restricciones que se asemeja a una técnica para optimización multiobjetivo. Esta técnica usa un “filtro” el cual impone el concepto de dominancia de un conjunto de soluciones parcialmente ordenado.

Desde una perspectiva híbrida-paralela, Lee et al. [105], proponen enfoque basado en dos AEs (Programas Evolutivos) paralelos híbridos denominados FGPEA y CGPEA, de grano fino y grueso respectivamente. Cada uno de ellos trabajando en diferentes niveles pero en forma cooperativa de manera de establecer un balance entre exploración y explotación. Dicho balance es controlado por el porcentaje de migrantes y la migración de los mejores individuos. En esta arquitectura el porcentaje de migración es alto en el nivel bajo (Grano fino) y bajo en el nivel más alto (Grano grueso). Por otro lado, la diversidad es mantenida dividiendo a los individuos en varios grupos, junto con una política de migración de individuos entre dichos grupos. Los resultados de la simulación reportados indican que la estructura híbrida paralela grano fino-grueso supera a los enfoques seriales, paralelos de grano fino y grueso respectivamente. El método fue estudiado sobre cinco de los problemas propuestos por Michalewicz [118].

Wah et al. [172, 171] presentan un marco que es independiente del problema, el cual unifica varios mecanismos (Simulated Annealing, Búsqueda greedy y Algoritmos Genéticos) para resolver problemas de programación no lineal discretos cuyas funciones objetivos no necesariamente son continuas y diferenciables. Dicho marco está basado en condiciones necesarias y suficientes de la teoría de optimización combinatoria para problemas restringidos usando multiplicadores de Lagrange. La búsqueda está dirigida a puntos de *ensilladura* de un vecindario discreto de manera que realiza *descensos* en el subespacio de original y *ascensos* en el subespacio de multiplicadores de Lagrange (ver figura 4.7).

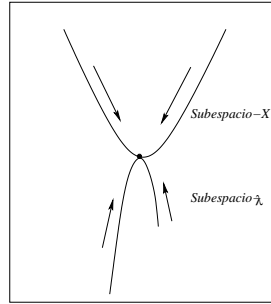


Figura 4.7: El problema se divide en dos subespacios y se trata de alcanzar desde cada uno de ellos el punto de *ensilladura*.

El artículo propone tres algoritmos basados en el marco previamente descrito: CSA [173] (Constrained Simulated Annealing), CGA (Constrained Genetic Algorithm) y CSAGA (Constrained Simulated Annealing + Genetic Algorithm). El algoritmo CSA realiza una búsqueda del punto de ensilladura a través de ascensos y descensos probabilísticos en el subespacio de los multiplicadores de Lagrange y el subespacio original del problema respectivamente. La probabilidad de aceptación de nuevos puntos está regida por Metrópolis. El segundo algoritmo, CGA, organiza la búsqueda del punto de ensilladura evolucionando una población de puntos candidatos según su adaptabilidad en el espacio Lagrangiano. Por su parte, el algoritmo CSAGA integra los dos algoritmos previos en un proceso combinado. En una primera fase dentro un proceso iterativo, cada individuo de la población es modificado vía el esquema seguido en CSA. La población resultante es evolucionada por un cierto número de generaciones. Dichas fases se repiten durante un determinado número de iteraciones. Los algoritmos fueron aplicados al conjunto de problemas $G01 - G10$ [118, 104]. Los resultados muestran una superioridad de los métodos CSA, CGA y CSAGA en relación a aquellos resultados reportados por diversos enfoques evolutivos para manejo de restricciones. Asimismo se concluye la superioridad de CSAGA, es decir el método combinado, por sobre los otros dos, CSA y CGA respectivamente.

El trabajo de Le Riche et al. [150], propone una nueva visión para una función de penalidad adaptativa basada en la resolución del problema de optimización dual. La solución encontrada (λ) en el espacio de los multiplicadores de Lagrange es usada para definir una función de penalidad exacta. El enfoque apunta a la construcción de penalidad mínima en comparación con enfoques similares. El método consta de un algoritmo evolutivo cuya función de penalización no contiene parámetros y cuyas propiedades garantizan optimalidad global. El algoritmo evolutivo itera entre el espacio primal y el espacio del problema dual aproximado. Dicho método fue aplicado a tres problemas para los cuales no hay garantía de la existencia de puntos de ensilladura, lo que los hace más difíciles para un método basado en Lagrange.

Parmee et al. [132] describen una metodología de búsqueda genética para ser usada en el contexto de problemas de ingeniería. Esta metodología permite establecer en una primera etapa un punto factible específico dentro del espacio de diseño complejo. A partir de este punto (encontrado a través del sistema VEGA [156]) se establece alrededor del mismo un vecindario de búsqueda para realizar la exploración. Dicho vecindario está representado por un hipercubo el cual es el resultado de aplicar un AG (segunda etapa) a un conjunto de posibles hipercubos cuya calidad se mide en función de los respectivos valores límites de cada variable en relación a las restricciones violadas. En una tercer etapa, se realiza una búsqueda local a partir del punto encontrado por VEGA y acotado a la región establecida por el hipercubo obtenido en la segunda etapa. El propósito de este método es evitar el uso de funciones penalidad las que en general producen importantes distorsiones en el espacio búsqueda original, lo que dificulta la obtención de soluciones factibles y de alta calidad.

En el ámbito de Particle Swarm Optimization (PSO) [97] existen algunas aplicaciones de este método a problemas de optimización numérica. Parsopoulos et al. [133] han propuesto recientemente la incorpora-

ción a PSO, de un enfoque de función de asignación de penalidades con múltiples-etapas y no estacionaria. Tres variantes de PSO con este enfoque de penalidad son consideradas: PSO-IN (*inertia weight*), PSO-Co (*constriction factor*), y PSO-Bo (combinación de los dos anteriores). Las tres alternativas fueron estudiadas sobre un conjunto de seis problemas bien conocidos. Los resultados globales muestran un comportamiento satisfactorio de PSO en todas las variantes presentadas. Sin embargo, los autores sostienen que el método puede ser mejorado usando operadores especializados para lograr la factibilidad de las partículas; y además, estudiar el método en otros problemas conocidos no estudiados aquí.

4.3.2. Técnicas de manejo de restricciones en algoritmos ACO

El enfoque ACO (ver Sección 3.1) fue desarrollado en sus inicios para problemas discretos donde el orden de las componentes en la solución es importante, por ejemplo TSP, el Problema de Asignación Cuadrática, entre otros). La pregunta es: ¿cómo podría ser extendido este método a problemas (con o sin restricciones) en el cual deba ser considerado un espacio continuo?, o ¿cómo podría ser extendido dicho enfoque para problemas combinatorios con restricciones en donde el orden de las componentes en la solución no es relevante?.

En el caso de los espacios continuos, algún tipo de discretización es necesaria para poder aplicar algún algoritmo ACO. Bilchev et al. [22] propusieron una estructura discreta la cual representa un conjunto finito de posibles direcciones en el espacio de soluciones hacia donde dirigir la búsqueda por un agente particular. Siguiendo la idea principal del enfoque ACO, el rastro de feromona es depositado sobre las direcciones que un agente puede elegir para realizar la búsqueda. En cada iteración del algoritmo las hormigas son enviadas en diferentes direcciones. La decisión con respecto a la dirección a seguir para cada hormiga está basada en la proporción de rastro acumulado en cada una de las posibles direcciones. Una vez que una dirección es escogida, cada agente realiza un

procedimiento de búsqueda local dentro de un radio de exploración. Las pocas experiencias de la aplicación del enfoque ACO a problemas de optimización numérica pueden ser encontrados en [22, 23, 21]. Los casos de prueba considerados incluyen problemas sin restricciones (El conjunto de funciones de prueba de De Jong) y varios problemas restringidos: cinco casos propuestos por Michalewicz [113], la función de Keane [96] y algunos problemas de optimización de diseño ingenieril. Básicamente el método para manejo de restricciones usado para resolver estos problemas, está basado en la utilización de funciones de penalización. Por ejemplo, en [23] es propuesto un modelo muy simple: la violación de la restricción (medida como la distancia Euclidiana de la región factible) determina la aceptabilidad de un determinado punto del espacio de búsqueda. El grado de violación de una restricción es implementada como una función lineal de la razón entre la región factible y el espacio de búsqueda total (o su aproximación). Luego, mientras más pequeña es esta razón, menor es la pendiente. Una descripción detallada de algoritmos ACO para espacios continuos es mostrada en el capítulo 7.

Otros trabajos realizados por Lei et. al. [108, 109] adaptan el concepto de ACO para ser aplicado a espacios continuos de la siguiente manera. El espacio de búsqueda es dividido en n subregiones, es decir la discretización, similarmente a la propuesta de Bilchev, es sobre el espacio de búsqueda. Es importante aclarar que Lei et al. consideraron casos de prueba multimodales sin restricción y un dominio simple de una variable real. En consecuencia, la división en n subregiones, corresponde a una división de n subintervalos del eje de las abscisas. Inicialmente, en el punto medio de cada subintervalo es depositada una hormiga la que en principio, se encarga de realizar la búsqueda acotada a dicho intervalo asignado. A medida que la búsqueda avanza, cada una de las hormigas en forma independiente, realizará un desplazamiento dentro del intervalo asignado de acuerdo a la calidad de la solución encontrada. Eventualmente habrá regiones solapadas de búsqueda comunes a algunas hormigas,

dado que en ciertas subregiones puede haber puntos de atracción (óptimo local o global). El manejo de la distribución del rastro de feromona sobre el intervalo i está dado por una función T_i unimodal (tipo campana) cuyo pico variará según la calidad de la solución x_i , la cual se encuentra en el centro de intervalo i . Es a través de esta función que el algoritmo se retro-alimenta a fin de aprender cuáles son las mejores sub-regiones a explorar. Existen parámetros adicionales que controlan la persistencia y evaporación del rastro. Luego, la decisión sobre cuál subintervalo depositan las hormigas de la colonia, será proporcional al monto de rastro sobre cada intervalo. El trabajo de Lei et. al. [108, 109], está limitado a unos pocos problemas continuos sin restricciones, aunque dicho enfoque poría ser la base para un método que incluya el manejo de restricciones de distintos tipos. Cabe destacar que la propuesta presentada por los autores asume que previo a la aplicación del algoritmo, se ha realizado una búsqueda usando un AG u otro método. De esta manera, el algoritmo ACO es concebido para actuar en una fase complementaria en la parte final de un proceso de búsqueda combinado. Es decir que una vez que en la primera etapa se ha realizado una exploración global del espacio de búsqueda y se han detectado zonas promisorias del mismo, el algoritmo ACO se encarga de realizar una búsqueda en dichas zonas. Sin embargo, la propuesta en esta tesis para el enfoque ACO en espacios continuos incorpora la búsqueda en la frontera y el mismo es el único encargado de realizar la exploración del espacio del problema.

El enfoque ACO también ha sido aplicado al CSP (ver Luk et al. [159]). El algoritmo ACO, más específicamente un SH, es comparado con varios algoritmos representativos el estado del arte en el campo de computación evolutiva para este problema. El SH propuesto por Luk et al. sigue la nomenclatura de Dorigo et al. [39] en cuanto a la representación de los problemas a resolver: una instancia del problema es visualizada como un grafo en el cual cada vértice representa una variable y cada arco o conexión entre los vértices, representa la restricción entre las respecti-

vas variables. Los resultados obtenidos, a partir de la aplicación del SH comparados con otros enfoques evolutivos, muestran que el SH fue superior al resto en términos de velocidad y porcentaje de convergencia. Los autores sostienen que el enfoque ACO resultó muy adecuado para CSP y que éste podría ser extendido a través de la inclusión de penalidades, entre otras propuestas.

Siguiendo en el ámbito de espacios continuos, Dreio et al. [51] proponen una nueva formalización para el diseño de algoritmos ACO para lo cual introducen el concepto de *heterarquía*¹⁰ y canales de comunicación. Si bien la aplicación del método sólo considera un caso de prueba (función continua multimodal sin restricciones) es interesante para ser destacado en esta sección. El diseño de este algoritmo se basa en la idea que la comunicación indirecta a través del rastro de feromona no es necesariamente el único medio que se puede explotar. Por ejemplo, la noción del concepto de *heterarquía densa* como explicación del comportamiento de ciertos insectos, da lugar a un algoritmo *heterárquico* el cual aprovecha las ventajas de cierto flujo de información que pasa a través de los agentes. Dicha información es intercambiada usando *canales de comunicación* con propiedades bien definidas, lo cuales pueden ser directos y/o indirectos. El canal indirecto está representado por el clásico depósito de rastro de feromona aunque de manera diferente a la usual. En este caso se generan en ciertos puntos del espacio de búsqueda, montículos de feromona cuyo tamaño depende de la calidad de la solución encontrada. Dichos montículos pueden ser percibidos por todos los miembros de la colonia y al mismo tiempo difundirlos en el espacio de búsqueda. Cada agente es atraído hacia ciertos montículos según la distancia que los separe y el tamaño del montículo. Aquí se plantea que este canal de comunicación es similar al concepto usado por los métodos PSO [97] y “path-relinking” [75]. El canal de comunicación directo establece la posibilidad de envío de mensajes entre los agentes y cada uno de ellos

¹⁰ En Inglés *heterarchy*

mantiene los mensajes recibidos durante un cierto tiempo. Los mensajes incluyen la posición del agente (punto en el espacio de búsqueda) y el valor de la función objetivo. De esta manera cada agente puede decidir moverse hacia otras regiones del espacio de búsqueda de acuerdo a la información contenida en los mensajes recibidos. Lo interesante de la propuesta es que el algoritmo puede trabajar usando dos canales complementarios de comunicación: directa, la cual permite la intensificación o explotación indirecta, la cual permite la diversificación o exploración de acuerdo a soluciones previamente evaluadas. En esta línea de investigación, Monmarché et al. [122], en un trabajo previo al anteriormente descrito de Dréo et al. [51], presentan un algoritmo ACO (denominado API), el cual consiste de un proceso de búsquedas paralelas en el espacio de soluciones estableciendo *sitios de caza* (puntos en el espacio de búsqueda) según la respectiva calidad de los mismos. Estos sitios pueden trasladarse a lo largo de la ejecución del algoritmo (exploración). El traslado de los sitios es realizado después de la aplicación de búsqueda local alrededor de dicho punto. El algoritmo API fue aplicado a un conjunto de casos de prueba conocidos (funciones continuas sin restricciones) dando resultados prometedores aunque una de las críticas que recibe este método (ver [51]) es que hace un uso pobre de la experiencia pasada para guiar el proceso de búsqueda.

Desde sus inicios, el enfoque ACO ha sido aplicado a problemas de optimización combinatoria que como se mencionó al comienzo de esta sección, tienen la característica que una solución a dichos problemas puede ser representada como una permutación de enteros. Estos problemas (TSP, QAP, etc.) sin bien, incluyen restricciones, las mismas no necesitan ser manejadas en forma explícita. Por ejemplo, una solución para TSP deber ser un camino Hamiltoniano desde la ciudad inicial a la final, pasando por todas la ciudades que son parte del problema. Los algoritmos ACO logran soluciones con estas características usando la estructura llamada *tabu*, la cual representa la memoria del agente, evitando

así la repetición de ciudades en la solución. Esto implica que el algoritmo “explora” sobre un espacio de permutaciones. Una situación similar se puede observar para otros problemas de optimización combinatoria. Sin embargo, existen problemas para los cuales existen restricciones explícitas las cuales crean regiones no factibles en el espacio de soluciones del problema. Ejemplos de estos problemas son MKP, SCP y MISP, entre muchos otros, los cuales son objeto de estudio de esta tesis. Todos estos problemas tienen las siguientes características: tienen restricciones dadas explícitamente, el espacio de soluciones factibles es un subconjunto del espacio total, y cada solución factible incluye un subconjunto propio de las componentes del problema (en el siguiente capítulo se presenta la propuesta desde la perspectiva de los AEs y de los algoritmos ACO). En relación a este tipo de problemas en el contexto de la metaheurística ACO, podemos mencionar los trabajos de Leguizamón et al. [106, 66] los cuales presentan las bases para una reformulación general de un algoritmo ACO (SH) para el problema de mochilas múltiples (MKP)¹¹ y conjunto independiente máximo (MISP)¹². Con respecto a SCP, existe un artículo no publicado del año 1999, que junto con [106, 66] son referenciados en Dorigo et al. [49], como las primeras aplicaciones de este método a los mencionados problemas. En el caso de SCP, también se menciona en [49] el artículo de Hadji et al. [137] como otra de las propuestas para resolver este problema. Otro artículo, Silva et al. [162] propone un SH para SCP. Si bien la propuesta es muy similar y posterior a la nuestra, el estudio de la aplicación se limita a un solo caso de prueba muy pequeño a partir del cual es evaluado el impacto en el comportamiento del algoritmo de ciertos parámetros del mismo.

¹¹El algoritmo SH para MKP es destacado en un artículo de Dorigo et al. [27] sobre aplicaciones relevantes de algoritmos ACO hasta ese momento.

¹²Resultados experimentales serán analizados en capítulos posteriores.

Capítulo 5

Algoritmos basados en el enfoque de frontera

5.1. Introducción

El manejo de restricciones, principal tema de esta tesis, fue desarrollado en el capítulo anterior a través de una revisión de los métodos más conocidos para su tratamiento. En este capítulo se destaca la importancia y conveniencia de definir métodos para búsqueda de soluciones en la frontera entre el espacio de soluciones factibles y no factibles para un problema determinado.

Es importante destacar que la búsqueda en la frontera es considerada como una forma alternativa de búsqueda para resolver problemas con restricciones, cuya aplicabilidad depende de factores como las características del problema, como así también de resultados de experiencias que muestran que para muchos problemas la mejor solución encontrada se encuentra sobre la frontera entre el espacio factible y no factible donde al menos una de las restricciones es activa.

En el capítulo anterior se establecieron algunas diferencias en cuanto a la forma en que cada uno de los enfoques estudiados aquí realiza la búsqueda en el espacio de soluciones. Es claro que la consideración de problemas discretos y continuos para la aplicación del enfoque de frontera variará según la metaheurística usada, sea ésta AEs o AHs. Por

ejemplo, para problemas discretos y continuos, los AEs aplican operadores que actúan sobre la codificación de una solución completa; dicha solución puede estar representada por una cadena de bits o un vector de punto flotante. En el caso de los AHs, la situación es un tanto diferente ya que existe una diferencia entre su aplicación a problemas continuos y discretos. Para el caso continuo, un AH es comparable a un AE en relación a la representación de las posibles soluciones y la manera en que operan sobre tales soluciones. Sin embargo, cuando el problema es de tipo discreto (asumimos aquí problemas restringidos), la solución se construye paso a paso y en ese proceso, para ciertos problemas de optimización combinatoria, podemos decir que genera soluciones sobre la frontera. Es decir, un AH para problemas discretos puede ser visualizado como un ‘algoritmo de frontera’ por la manera en que las soluciones son construídas.

En síntesis, el presente capítulo resume las propuestas generales para manejo de restricciones basado en un enfoque de frontera que considera, por un lado, la extensión del enfoque de frontera para problemas continuos en el ámbito de AEs y AHs. Por otro lado, una parte importante de la propuesta global, es el hecho de que el enfoque ACO fue propuesto originalmente para problemas que pudieran ser representados naturalmente como un grafo y una solución al mismo, como una permutación de enteros, es decir un ordenamiento de las componentes del problema. Sin embargo, dicho enfoque puede ser adaptado para otros tipos de problemas, como se propone aquí, donde el orden de las componentes no es lo importante sino el subconjunto de componentes que conforman la solución. Para el caso de los AEs, se presenta un enfoque que involucra la aplicación de operadores especializados que combinan la preservación de las similitudes entre los individuos, con la aplicación de procesos iterativos para llevar la solución hacia la frontera entre el espacio factible y no factible.

La parte final de este capítulo presenta las propuestas de manejo de

restricciones para las metaheurísticas consideradas y su aplicabilidad a espacios de búsqueda discretos y continuos. El capítulo concluye con la presentación de los problemas considerados en el estudio experimental.

5.2. Importancia de la búsqueda en la frontera

Una de las principales razones en relación a las dificultades en alcanzar soluciones globales, está vinculada con la incapacidad de muchos métodos de explorar entre la frontera del espacio búsqueda factible y no factible. La posibilidad de llevar a cabo de manera eficaz este tipo de búsqueda puede ser muy importante, dado que para muchos problemas de optimización no lineales restringidos, al menos algunas restricciones se encuentran activas en el óptimo global [118, 114, 121].

Es claro que las soluciones óptimas yacen sobre el borde cuando las restricciones son lineales, ya sea para espacios de búsqueda convexos como para los no convexos. Pero cuando las restricciones no son lineales, es imposible garantizar que la solución óptima, o una cercana a la óptima, se encuentre sobre el borde entre el espacio factible y no factible. Sin embargo, cuando las restricciones del problema involucran al menos una restricción de igualdad, es decir una ecuación $h_j(\mathbf{x})$, es claro que independientemente de la convexidad del problema y del espacio de búsqueda, la solución óptima se encontrará en el borde. Además, existe evidencia que soporta el desarrollo de metaheurísticas que exploren entre la frontera del espacio de búsqueda factible y no factible. Michalewicz et al. [118, 114, 121] han reportado resultados de varios experimentos con problemas de optimización numérica para las cuales las restricciones se encuentran activas en el óptimo o en la mejor solución conocida hasta el momento. Pardalos et al. [58] presentó una colección de problemas de optimización numérica con diversas características incluyendo varios casos de prueba no lineales para los cuales la solución óptima o mejor conocida produce algunas restricciones activas del problema, es decir, que

la solución descansa sobre el borde de al menos una de las restricciones del problema.

Por otro lado, es importante remarcar la importancia de la búsqueda en la frontera en problemas de optimización combinatoria. La figura 5.1 muestra un hipercubo de dimensión 4 el cual representa un espacio de búsqueda de un problema hipotético de optimización combinatoria restringido con restricciones del tipo $g_j(\mathbf{x}) \leq 0$. Supongamos que una solución es factible cuando incluye a lo sumo dos componentes del problema. Para nuestro problema, el conjunto de soluciones factibles es $\mathcal{F} = \{0000, 0001, 0010, 0100, 1000, 0011, 0101, 0110, 1001, 1010, 1100\}$. Sin embargo, el conjunto frontera está representado por un subconjunto de \mathcal{F} , es decir, aquellas soluciones que involucran dos componentes (aparecen subrayadas en la figura 5.1): $\{0011, 0101, 0110, 1001, 1010, 1100\}$. Los puntos restantes, representados por el conjunto $\mathcal{U} = \{0111, 1011, 1101, 1110\}$ conforman el espacio de búsqueda no factible. Por lo tanto, para este tipo de problemas combinatorios el beneficio de desarrollar operadores de frontera (borde) son similares a aquellos problemas de optimización numérica de acuerdo a las características mencionadas previamente, es decir, limitar exploración de las posibles soluciones a la frontera y eventualmente prescindir del uso de algoritmos de reparación o funciones de penalización, aunque no siempre es posible.

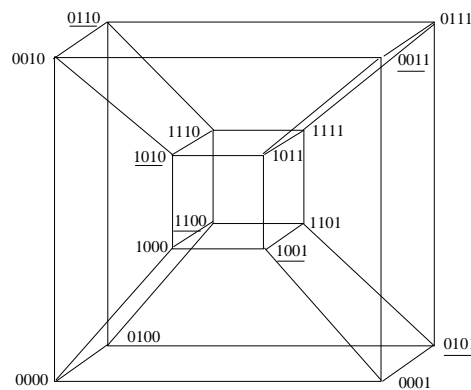


Figura 5.1: Hipercubo representando un espacio de búsqueda hipotético formado por cadenas binarias

Es importante destacar que desde el ámbito de estudio de distintas metaheurísticas, se ha reconocido la necesidad de realizar la búsqueda en áreas cercanas a la frontera de la región factible. Por ejemplo, uno de los desarrollos más recientes para optimización restringida es el llamado método de *oscilación estratégica* que fue introducido a través de Scatter Search (Glover [71]). Su aplicación incluye un variedad importante de problemas de optimización combinatoria y no lineal (ver Glover et al. [74]). La técnica de oscilación estratégica está basada en la identificación de niveles críticos que representan la frontera entre la región factible y no factible. La estrategia básica de este enfoque es identificar ciertas regiones de búsqueda e inducir un patrón de búsqueda para visitar dichas regiones a distintas profundidades dentro de los límites. Así, dentro de un patrón variable, el método se aproxima y aleja de la frontera a la manera de ondas oscilantes. Ejemplos de tales regiones (ver Corne et al. [39]) y su frontera asociada pueden verse en la tabla 5.1.

Regiones	Frontera
1. Factible y No factible	Determinada por las restricciones
2. Construcciones parciales o excesivas	Una construcción completa (knapsack, set covering, clique, etc.)
3. Schedules incompletos o saturados	Schedules adecuados
4. Optimos o subóptimos	Soluciones sin mejora inmediata
5. Agrupamientos de soluciones superiores	Regiones entre los agrupamientos
6. Vecindarios alternativos	Transiciones según los tipos de movimientos

Tabla 5.1: Diferentes *regiones* en el espacio de búsqueda y las fronteras asociadas.

Existen variantes en cuanto a las regiones y las fronteras asociadas. Por ejemplo, se podría reemplazar la dicotomía entre factible y no factible por un método que se enfoque en restricciones críticas que permitan

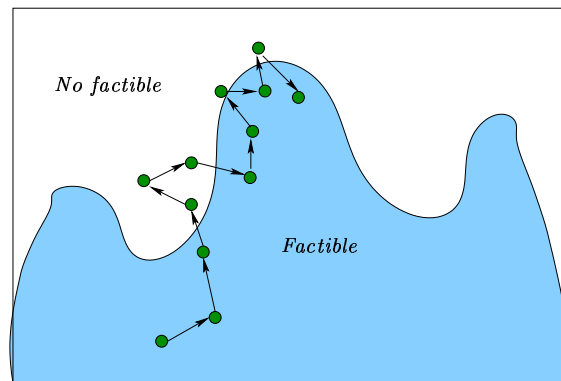


Figura 5.2: Oscilación estratégica entre la frontera del espacio factible y no factible.

definir dominios de “no factibilidad parcial”. En cada caso, el enfoque de oscilación estratégica opera moviéndose en una región hacia la frontera, y luego ya sea, cruza la frontera o se vuelve para seguir en la misma región recientemente recorrida (es el caso de la oscilación por un “solo lado”).

Las técnicas metaheurísticas tienen un alto potencial con respecto a la incorporación o adaptación de alguna de sus componentes a fin de explorar en la frontera de espacio factible y no factible. Por ejemplo, operadores de crossover y mutación especializados que podrían ser diseñados para explorar exclusivamente en la frontera, es decir, “operadores frontezos” de manera tal que el espacio factible sea cerrado bajo su aplicación e inicializar la población inicial con soluciones factibles sobre la frontera. Luego, todos los individuos potenciales que puedan ser creados vía la aplicación de dichos operadores especializados estarán sobre la frontera también. Inclusive un beneficio adicional puede ser obtenido si el proceso de búsqueda se restringe a un subconjunto del espacio de búsqueda total, dado que sólo la frontera es considerada.

De manera similar, el enfoque ACO tiene también un gran potencial para explorar solamente la frontera de la región factible del espacio de búsqueda para problemas restringidos. Como ejemplo se pueden mencionar los resultados reportados por Bilchev [24, 22] en los cuales diferentes problemas restringidos y no restringidos de optimización numérica fueron

considerados. La importancia de este enfoque (solamente para espacios continuos) es que la colonia está a cargo de evolucionar un conjunto de soluciones a través de la aplicación de búsqueda local de una manera iterativa. Por lo tanto, los operadores de búsqueda local que pueden ser aplicados aquí, son iguales o similares a los usados como operadores de mutación en un algoritmo evolutivo trabajando bajo el esquema de exploración sobre la frontera. Un análisis detallado será presentado en la sección 5.4 en relación a la aplicación del enfoque de frontera a problemas de optimización combinatoria usando algoritmos evolutivos y algoritmos basados en colonias de hormigas.

5.3. Desarrollos de investigación en búsqueda en la frontera

El artículo de Michalewicz et al. [120] representa uno de los primeros trabajos en búsqueda en la frontera a través de algoritmos evolutivos. La efectividad del enfoque para problemas de optimización numérica con restricciones es mostrado considerando dos problemas bien conocidos, la función de Keane (F_K) [96] y la función producto (F_P). Para el caso de F_K , se propone un crossover denominado *geométrico* y un operador especial de mutación. Ambos operadores generan hijos sobre la frontera entre el espacio factible y no factible. De manera similar, para el caso de F_P , un operador de crossover denominado *esfera* es diseñado de forma tal que solamente son generados los puntos de la esfera definidos por la restricción del problema. También se usa un operador de mutación especial que mantiene los hijos generados en la frontera determinada por dicha esfera. La figura 5.3 muestra una superficie hipotética de una frontera donde son aplicados operadores de crossover y mutación los cuales generarían hijos sobre la frontera. Para el caso del operador de crossover es tal que $\chi(p_1, p_2) = c_1$ y para el de mutación, $\psi(p_3) = c_2$, donde p_1 , p_2 , c_1 , p_3 y c_2 pertenecen a la frontera entre el espacio factible

y no factible.

Una comparación del enfoque de frontera contra el crossover aritmético demuestra un incremento significativo en la calidad de los resultados del algoritmo evolutivo (basado en representación de punto flotante). Además, este enfoque puede ser generalizado en el siguiente sentido [120]: Experimentos iniciales (usando cualquier método de optimización) podrían indicar las restricciones activas para un problema en particular. A partir de estos resultados, sería posible diseñar un algoritmo con operadores especializados que limiten la búsqueda a la frontera entre el espacio factible y no factible con la eventual mejora en la calidad de las soluciones encontradas.

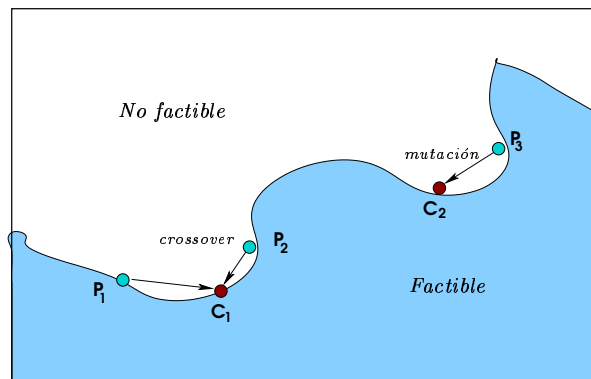


Figura 5.3: Operadores genéticos que dan como resultado hijos en la frontera.

Schoenauer et al. [157] proponen una variedad de operadores evolutivos teóricos que pueden explorar una superficie general de dimensión $n - 1$ (n es el número de variables del problema). Estos operadores dependen de la representación de la superficie: basada en curvas, representación paramétrica y operadores basados en planos. Algunos resultados experimentales para esta propuesta han sido reportados por Schoenauer et al. en [121] para los siguientes tres casos: función producto (F_P) y dos funciones adicionales, B_1 y B_2 , las cuales son una versión restringida de la función original (no restringida) propuesta por [14]. Estos nuevos operadores basados en una representación general de la superficie fueron probados en dos enfoques evolutivos: AGs y EEs. También

fueron incluidos los operadores de frontera originales [120], crossover esfera y la mutación y operadores que incluían una fase de reparación. Los resultados de los experimentos indican que los operadores muestran una performance diferente según las funciones consideradas. Sin embargo, para los tres problemas considerados, el enfoque de frontera original (el cual limita la búsqueda a la frontera) propuesto por Michalewicz et al. [120] fue superior o tan bueno como aquellos operadores que consideran una representación general de la superficie. Además, es importante remarcar la importancia del operador esfera para la función F_P . Para este ejemplo, el AG obtuvo los mejores resultados usando una probabilidad de aplicación alta y baja para los operadores de crossover y mutación respectivamente, mientras que la EE, la cual sólo usa mutación para la exploración, obtuvo los peores resultados. Sin embargo, la situación cambió completamente para las funciones B_1 y B_2 . La EE mostró la mejor performance. El AG para estos problemas sólo mejoró su performance usando un valor máximo de probabilidad para la aplicación de la mutación ($p_m = 1$), lo que muestra la importancia del operador en los experimentos realizados.

Una de las principales ventajas de este método tiene que ver con la reducción del espacio de búsqueda, dado que la exploración se realiza solamente sobre la frontera de la región factible. Estudios sobre la aplicación de este tipo de enfoque han mostrado una mejora importante en el performance de los AEs para algunos casos como los mencionados arriba, aunque dichos casos de testeo conforman funciones que sólo tienen una restricción o bien una sola de ellas se encuentra activa en el óptimo y por consiguiente se han diseñado operadores especializados (por ejemplo *geométrico* y *esfera*) que se ajustan perfectamente a la frontera definida por las respectivas restricciones de manera que el conjunto frontera es cerrado bajo su aplicación. Sin embargo, esto puede transformarse en una seria desventaja ya que no siempre es posible definir operadores “*ad hoc*” para cada caso y por otro lado, podría ser necesario definir más de

un operador si quisiéramos extender la aplicación de este enfoque para problemas con más de una restricción. En este sentido, se propone un enfoque de frontera más general, lo que no significa “un enfoque general”, en el cual no necesariamente se defina un operador especializado según la restricción, sino que a través de un esquema general, un operador pueda ser útil para distintas restricciones, sin importar la forma o complejidad de la misma.

En un artículo reciente de Wu et al. [177] se presenta un AG aplicado a la optimización de un sistema de distribución de agua, un problema altamente restringido. En enfoque propuesto co-evoluciona y auto-adapta los factores de penalización que manera tal que el AG dirija y preserve la búsqueda hacia la frontera entre el espacio factible y no factible. Una de las principales ventajas es la reducción de las evaluaciones necesarias para alcanzar soluciones óptimas o cercanas. En este artículo se destaca principalmente la situación que para muchos problemas con funciones objetivo y restricciones no lineales, el óptimo está sobre la frontera (o bien, muchas, sino todas, de las restricciones del problema están activas en ese punto). Otro punto importante es que en la descripción de los antecedentes, Wu et al. mencionan algunos pocos trabajos relacionados con la búsqueda en la frontera y particularmente, resalta la imposibilidad de diseñar operadores de frontera (como aquellos propuestos por Schoenauer et al. [157]) usando un enfoque analítico dadas las características generalmente encontradas en este tipo de problemas, es decir, restricciones altamente no lineales. Sin embargo, esta situación podría ser superada de cierta manera, usando el método propuesto en esta tesis para la búsqueda en la frontera para este tipo de problemas (ver más adelante en éste y los siguientes capítulos relacionados).

Por otro lado, los artículos de Gottlieb [80, 78, 81] y Gottlieb et al. [79] son los más representativos de búsqueda en la frontera en el ámbito de AEs aplicados a problemas de optimización combinatoria. El estudio está destinado a la resolución de MKP y en ellos se refleja la importan-

cia de la búsqueda dirigida hacia la frontera desde distintas perspectivas: uso de decodificadores, penalización y reparación. Uno de estos trabajos fue comentado en la sección 4.3.1 en relación al uso de métodos basados en decodificadores. En [81] se estudia el uso de la penalización y las propiedades que una función de penalidad debería poseer para evitar lo que se denomina “*problema de factibilidad*”¹. En esta dirección, se propone una función con un sesgo que dirige la búsqueda hacia la frontera entre la región factible y la no factible. Como conclusión de este estudio, se plantea que el problema de factibilidad puede ser resuelto si i) la función de penalidad incluye un sesgo que dirige la búsqueda hacia la zona factible y ii) la población inicial contiene al menos una solución factible y todas las soluciones no factibles deberán tener valores de *fitness* menores a cualquier solución factible. Esta situación no sólo se plantea para MKP, sino para todos los problemas de tipo cobertura y empaquetamiento² dado que una posible solución óptima para ellos se encuentra sobre la frontera entre la zona factible y no factible del espacio de búsqueda. Similares conclusiones son alcanzadas en [80] en donde se comparan distintos AEs que realizan la exploración: sobre todo el espacio de búsqueda; sólo sobre la región factible; o sobre el subconjunto de la región factible que se encuentra exclusivamente sobre la frontera. Una de las principales conclusiones alcanzadas establece que la búsqueda acotada a la frontera no garantiza por sí misma el mejor comportamiento de un AE, sin embargo, cuando es combinada con métodos avanzados de inicialización y/o búsqueda local, muestra resultados superiores a otros enfoques evolutivos para manejo de restricciones.

5.4. Enfoque propuesto

El principal objetivo de esta tesis es extender la aplicación del enfoque de frontera a problemas de optimización restringidos involucrando

¹La población final sólo contiene soluciones no factibles.

²*Covering* y *packaging* respectivamente.

variables discretas y continuas. Para el caso continuo, diferentes problemas de optimización numérica serán considerados con variaciones en el número y tipo de las restricciones. Por otro lado, para el caso discreto, el estudio se enfocará en problemas de subconjunto, es decir, aquellos problemas para los cuales la solución es un subconjunto de las componentes del problema y donde el orden en que aparezcan en la solución no tiene importancia.

5.4.1. Tipos de problemas considerados

En esta sección se describen algunas características generales en relación a la región de la frontera en espacios continuos y discretos. Las consideraciones son hechas en relación a la forma en la cual debería proceder la exploración en espacios continuos y discretos. En la próxima sección se explica en forma específica el enfoque de frontera según la perspectiva de los AEs y AHs.

Espacios continuos:

Basado en la formulación general de la programación no lineal, diferentes tipos de instancias pueden ser definidas según el número y tipo de restricciones. Además, si aquellos problemas tienen restricciones activas (ver concepto en sección 1.1.1) en el punto óptimo o mejor conocido o el conjunto de restricciones del problema incluye al menos una restricción por igualdad, las posibilidades de usar un enfoque de frontera se ven incrementadas. Nuestra propuesta consiste en definir operadores de exploración sobre la frontera para cada restricción en la formulación del problema. Si el problema tiene sólo una restricción, el método es directo y se puede proceder usando ya sea AEs o AHs para buscar en la frontera usando un operador especializado. Luego, si la solución está en la frontera, el algoritmo a ser aplicado eventualmente encontrará la solución óptima, dependiendo de la efectividad del algoritmo y del operador diseñado para tal fin.

La otra situación, y tal vez la más típica, es aquella en donde los problemas tienen más de una restricción: es decir, $g_j(\mathbf{x}) \leq 0$ para $j = 1, \dots, q$; y $h_j(\mathbf{x}) = 0$, para $j = q + 1, \dots, m$, con $m > 1$. Por lo tanto, la aplicación del enfoque de frontera no es tan directo como cuando se tiene una única restricción. Lamentablemente, no es posible la existencia de puntos que se sitúen al mismo tiempo sobre las distintas fronteras determinadas por cada una de las restricciones del problema, y menos aún, diseñar operadores de frontera que generen puntos que satisfagan dicha condición.

Desde el punto de vista del enfoque de frontera, las restricciones de desigualdad (inecuaciones) g_j pueden ser vistas como restricciones de igualdad (ecuaciones) dado que el enfoque dirige la búsqueda hacia la frontera, es decir, la región donde $g_j(\mathbf{x}) = 0$ para $j = 1, \dots, q$.

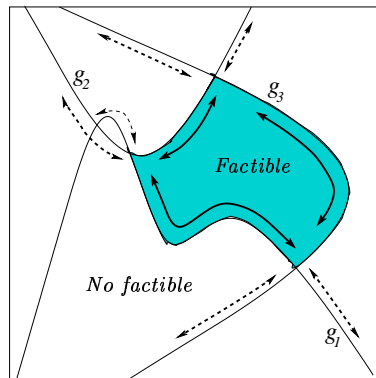


Figura 5.4: Espacio de búsqueda factible determinado por 3 desigualdades.

El enfoque de frontera propuesto para manejo de más de una restricción consiste en el diseño de un operador de frontera para cada restricción. Un posible algoritmo que use este enfoque procederá considerando cada restricción por separado. La selección de la restricción puede depender de resultados previos (es decir, tomando en cuenta las restricciones activas para el mejor punto conocido) o las características del problema (por ejemplo, y muy importante, las existencia de restricciones de igualdad).

La figura 5.4 muestra un espacio hipotético de búsqueda determina-

do por tres restricciones. Supongamos que la búsqueda procede sobre la frontera definida sobre la restricción g_1 usando el operador de frontera O_{g_1} (la línea llena en la figura 5.4). La aplicación de este operador producirá eventualmente soluciones que violen las restricciones g_2 y g_3 (líneas punteadas en la figura 5.4). Para soluciones sobre la restricción g_1 y que además violen las restricciones restantes, podría ser usado un factor de penalización incorporado a la función objetivo *eval*, aunque varios de los métodos descritos previamente (ver sección 4.3.1) podrían ser usados de manera similar.

$$eval(\mathbf{x}) = \begin{cases} f(\mathbf{x}) & \text{si ninguna restricción es violada} \\ f(\mathbf{x}) + p(\mathbf{x}) & \text{en otro caso} \end{cases} \quad (5.1)$$

En este sentido es importante destacar que la penalización no es parte del método en sí, sino que es una alternativa para diferenciar la calidad de las soluciones. Por ejemplo, otra posibilidad es medir la calidad relativa de las soluciones, independientemente de la restricción elegida, según el enfoque de coincidencia de patrones.

Dadas las características de los métodos poblacionales estudiados, es posible realizar un gran cantidad de variaciones en la manera en que se elige la restricción sobre la cual realizar la búsqueda. Una alternativa de aplicación directa es por ejemplo, que el algoritmo comience la búsqueda por una restricción en particular y que dependiendo de ciertas condiciones, cambie la restricción de referencia para hacer la búsqueda. Las condiciones para el cambio pueden ser tan simples como considerar un cierto número de iteraciones para cada restricción o algo más complejo, que puede depender del conocimiento y/o características del problema de manera tal de establecer prioridades sobre las restricciones a considerar. Existen otras posibilidades que pueden ser explotadas, según el método usado. Sin embargo, es necesario explicar en detalle la manera en que cada uno de estos métodos trabaja para poder visualizar con claridad algunas de tales posibilidades. Las estrategias aplicadas en

nuestro estudio serán explicadas en detalle en los respectivos capítulos relacionados al estudio experimental.

A través de este enfoque híbrido, se espera reducir significativamente el tamaño de la región del espacio de soluciones considerado durante el proceso de búsqueda dado que las únicas soluciones a tener en cuenta serán aquellas que estén sobre la frontera entre el espacio factible y no factible. El método para manejo de restricciones usado en este enfoque puede variar de acuerdo al problema, tipo de restricciones y la conveniencia de su aplicación en función de sus ventajas y desventajas inherentes. Sin embargo, este tipo de enfoque de frontera es aplicable si es posible definir un operador de frontera para cada una de las restricciones del problema, principalmente para aquellas que están activas en los mejores puntos conocidos.

Si bien la hipótesis para la aplicación del enfoque asume que es posible “diseñar” los operadores de frontera necesarios según el tipo de cada una de las restricciones del problema, puede ocurrir, y de hecho ocurrirá con una alta probabilidad, que las restricciones definan una curva de frontera para la cual no sea tan directo o posible diseñar tales operadores especializados como lo son los operadores de frontera.

Una alternativa que se propone aquí como un “operador generalizado” de frontera, se basa en que un punto \mathbf{b} sobre la frontera entre el espacio de búsqueda factible y no factible puede representarse por medio de dos puntos \mathbf{x} e \mathbf{y} donde \mathbf{x} es un punto factible e \mathbf{y} es no factible. Es decir que el par de puntos (\mathbf{x}, \mathbf{y}) representa un único punto (\mathbf{b}) sobre la frontera el cual es obtenido aplicando un sencillo proceso de búsqueda binaria para encontrar el punto de corte, sobre la curva que representa la frontera, del vector cuyos extremos está determinado por \mathbf{x} e \mathbf{y} respectivamente. La figura 5.5 muestra un espacio de búsqueda hipotético el cual se divide en la región factible y no factible. En dicha figura se muestran cuatro puntos sobre la frontera \mathbf{b}_1 , \mathbf{b}_2 , \mathbf{b}_3 y \mathbf{b}_4 , los cuales son obtenidos respectivamente a partir de los pares de puntos $(\mathbf{x}_1, \mathbf{y}_2)$, $(\mathbf{x}_2, \mathbf{y}_2)$, $(\mathbf{x}_3, \mathbf{y}_3)$

y $(\mathbf{x}_4, \mathbf{y}_4)$.

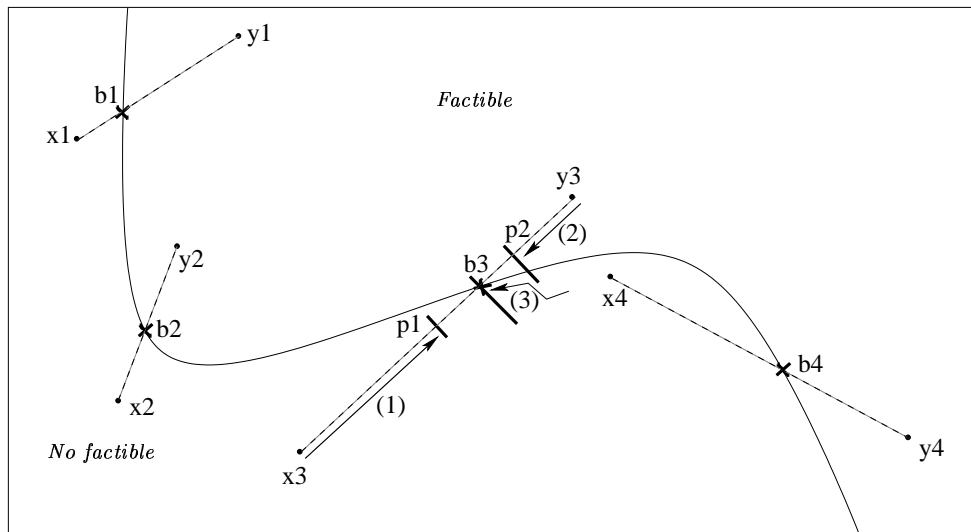


Figura 5.5: Con una sencilla búsqueda binaria se llega al punto de corte sobre la frontera.

La búsqueda binaria que se aplica a cada par de puntos para obtener el respectivo punto sobre la frontera se realiza según los pasos descritos en el algoritmo de la figura 5.6 y que puede visualizarse en la figura 5.5 cuando es aplicado al par de puntos $(\mathbf{x}_3, \mathbf{y}_3)$ de cual se obtiene el punto \mathbf{b}_3 . El primer paso rotulado con (1) indica que el límite inferior del rango de valores es corrido hacia el punto medio (\mathbf{p}_1) entre \mathbf{x}_3 e \mathbf{y}_3 debido a que \mathbf{p}_1 es no factible. En el próximo paso, se consideran los puntos \mathbf{p}_1 e \mathbf{y}_3 como extremos del vector. El punto medio obtenido es \mathbf{p}_2 , un punto factible. En consecuencia, el extremo factible del vector es corrido (2) de manera tal que coincida con el punto \mathbf{p}_2 . Finalmente, y a modo de ejemplo, el último punto generado es \mathbf{b}_3 , el cual está sobre la frontera o muy cercano a la misma, según lo controla la condición ($\text{dist_a_la_frontera}(\mathbf{m}) > \epsilon$) del algoritmo. Dicha condición establece un umbral sobre el número de iteraciones para alcanzar lo más cercanamente posible el punto de corte del vector cuyos puntos extremos son un punto factible y un punto no factible, respectivamente.

Es importante aclarar que el Algoritmo 5.6 actúa como un decodifi-


```

Algoritmo BB( $\mathbf{x}, \mathbf{y}$ : real vector): int
begin
  do
     $\mathbf{m} = \text{Punto\_entre}(\mathbf{x}, \mathbf{y})$ 
    if ( Frontera( $\mathbf{m}$ ) ) return  $\mathbf{m}$ ; /*  $\mathbf{m}$  es un punto de la frontera */
    if ( Factible( $\mathbf{m}$ ) )  $\mathbf{x} = \mathbf{m}$ ; else  $\mathbf{y} = \mathbf{m}$ ;
    while ( dist_a_la_frontera( $\mathbf{m}$ ) >  $\epsilon$  ); /* Lo más cercano posible,
                                          según  $\epsilon$  */
  return Punto_entre( $\mathbf{x}, \mathbf{y}$ );
end

```

Figura 5.6: Algoritmo que permite determinar el punto sobre la frontera dados dos puntos, uno factible y uno no factible.

cador que transforma los puntos (\mathbf{x}, \mathbf{y}) en uno y sólo un punto \mathbf{b} sobre la frontera y que el par de puntos que da origen al punto sobre la frontera es un argumento que sólo es modificado localmente como resultado del proceso de “decodificación”.

Hasta aquí se ha mostrado cómo un punto sobre la frontera podría ser representado usando un par de puntos cuyo segmento que los une, corte a la curva que representa a la frontera. La otra situación a considerar son los posibles operadores para la exploración del espacio de búsqueda. Desde la perspectiva de AEs, dichos operadores son el crossover y la mutación. En el caso de los AHs, operadores similares a la mutación pueden ser implementados para realizar la búsqueda. De esta manera podemos centrarnos en el crossover y la mutación para entender cómo puede ser explorado el espacio de búsqueda restringido a la frontera bajo este esquema generalizado. Para el caso del crossover, se toman dos pares de puntos; sean éstos $(\mathbf{x}_1, \mathbf{y}_1)$ y $(\mathbf{x}_2, \mathbf{y}_2)$ respectivamente, donde $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{F}$ y $\mathbf{y}_1, \mathbf{y}_2 \in \mathcal{U}$. Luego se aplica el tradicional crossover aritmético [114] entre los puntos \mathbf{x}_1 y \mathbf{x}_2 y los puntos \mathbf{y}_1 y \mathbf{y}_2 . Es decir, $\chi_a(\mathbf{x}_1, \mathbf{x}_2) =$

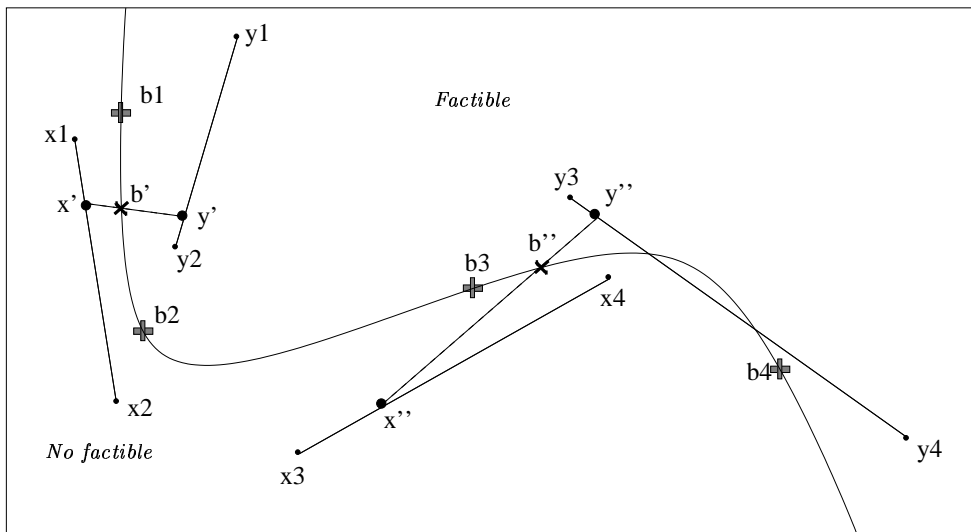


Figura 5.7: Crossover aritmético aplicado sobre dos pares de puntos para generar un nuevo punto sobre la frontera.

x' y $\chi_a(y_1, y_2) = y'$. La figura 5.7 muestra esta situación y para otra similar con los pares de puntos (x_3, y_3) y (x_4, y_4) . La aplicación del crossover aritmético no garantiza que el punto resultante sea factible (no factible) cuando los puntos padres son factibles (no factibles). Sin embargo, para garantizar que esto último ocurra, es necesario variar el parámetro α del crossover aritmético (puede ser en forma aleatoria) hasta que el punto generado esté sobre la región correspondiente, ya sea que se estén considerando puntos factibles o no factibles. Esta situación se puede visualizar en la figura 5.7 para los puntos x_4 e y_4 para los cuales un valor de α cercano a 0,5 daría como resultado un punto intermedio dentro de la región no factible, contrario a lo que se espera. Sin embargo, con un valor adecuado para α dicho crossover arrojará como resultado un punto dentro del espacio factible de acuerdo a los requerimientos del enfoque. Para el caso de la exploración gobernada por un operador tipo mutación, la filosofía es similar al crossover, la diferencia es que aquí es considerado sólo un par de puntos (x, y) y lo que se hace es modificar uno o ambos de los puntos del par con la salvedad que si un punto factible (no factible) es modificado, el nuevo punto deberá ser

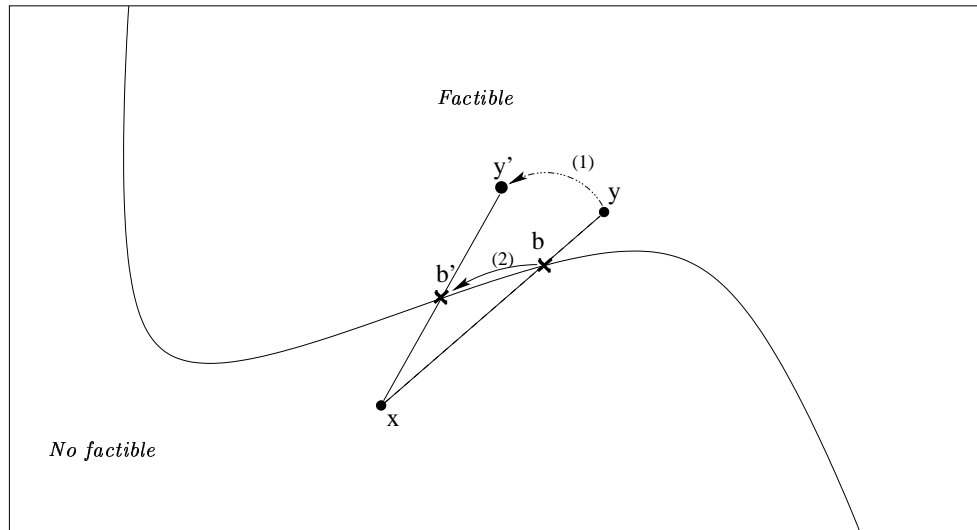


Figura 5.8: Una variación sobre uno o ambos puntos del par, permite cambiar el punto sobre la frontera.

factible (no factible). La figura 5.8 muestra el par de puntos (\mathbf{x}, \mathbf{y}) que representan el punto \mathbf{b} sobre la frontera. En este caso se asume que el punto \mathbf{y} ha sido modificado dando como resultado (1) un punto \mathbf{y}' en el espacio de soluciones factibles. Luego de la modificación, el nuevo punto de la frontera (2) representado por $(\mathbf{x}, \mathbf{y}')$ es \mathbf{b}' .

Espacios discretos.

Dado un problema de optimización combinatoria que incluye un conjunto de componentes $S = \{1, \dots, n\}$, $n \in \mathbb{N}$, decimos que P es un problema de subconjunto si una solución para P puede ser obtenida combinando los elementos de s , esto es, la solución al problema P es un elemento de 2^S . Sin embargo, si alguna restricción es incluida en la formulación del problema, el espacio de búsqueda quedará naturalmente dividido en una región factible \mathcal{F} y una no factible $\mathcal{U} = \mathcal{S} - \mathcal{F}$. Esta categoría involucra un cantidad muy importante de problemas combinatorios \mathcal{NP} -completos de gran importancia práctica: múltiples mochilas, cobertura de conjunto, máximo clique, y muchos otros [68].

En general, el espacio de búsqueda para estos problemas de optimización combinatoria puede ser representado con un conjunto $S = \{0, 1\}^n$,

donde n es el tamaño del problema. Sin embargo, el enfoque de frontera para manejar restricciones en estos problemas de optimización combinatoria depende de la metaheurística escogida para resolverlo, en nuestro caso un AE o AH. De cualquier manera, el enfoque propuesto para problemas discretos consiste en la manipulación y generación de soluciones que estén sobre la frontera según las características de los problemas discretos considerados en esta tesis (problemas involucrando restricciones del tipo, $g_j(\mathbf{x}) \leq 0$ o $g_j(\mathbf{x}) \geq 0$).

5.4.2. El enfoque de frontera

En esta sección se presentan las consideraciones generales para la implementación de la propuesta descrita en la sección anterior desde la perspectiva del enfoque evolutivo y basado en colonia de hormigas. Para los espacios continuos se da una descripción general considerando el diseño de los operadores de búsqueda, los cuales dependen en general del problema a ser considerado. Sin embargo, la propuesta para los problemas discretos (problemas de subconjunto) es dada como una formulación general para la cual serían necesarias sólo pequeñas modificaciones a fin de ser aplicadas a un problema particular.

5.5. Enfoque de frontera en AEs

Caso continuo:

Para el algoritmo evolutivo que implemente el enfoque de frontera, se deberán diseñar operadores adecuados de crossover y mutación que sean capaces de generar hijos que estén sobre la frontera. El diseño podría hacerse usando operadores específicos según la restricción, o bien, el método general explicado en el punto 5.4. Además, será necesario implementar algún método de manejo de restricciones (por ejemplo penalización) en caso de tener mas de una restricción según se explicó previamente. Sin

embargo pueden existir excepciones cuando las restricciones no están activas en la región de la restricción para la cual el operador ha sido diseñado. Por ejemplo, la función de Keane en [120] tiene dos restricciones y sólo una de ellas es activa en el mejor punto conocido, con lo cual la otra es descartada en el proceso de búsqueda limitado a la frontera.

Caso discreto:

En lo que sigue, se presenta un enfoque general para la aplicación de un algoritmo evolutivo para problemas de subconjunto con restricciones.

Uno de los primeros pasos en el proceso del diseño de un algoritmo evolutivo está relacionado con el tipo de representación a ser usada para las soluciones. El espacio de búsqueda para los problemas de subconjunto considerados aquí puede ser representado como un conjunto $S = \{0, 1\}^n$, es decir, el conjunto de todas las cadenas binarias de longitud $n \in \mathbb{N}$. Por lo tanto, una cadena binaria $x \in S$ representa una posible solución (factible o no factible) con el significado siguiente: $x[i] = 1$, luego la componente i está en la solución; en otro caso, $x[i] = 0$, esto es, la componente i no está en la solución.

Teniendo en cuenta esta representación, el objetivo es diseñar un algoritmo evolutivo que procese y genere solamente soluciones que estén sobre la frontera entre la región factible y no factible del espacio de búsqueda. A fin de satisfacer estos requerimientos críticos, las siguientes componentes al algoritmo evolutivo deberían ser diseñadas en forma acorde:

- **Población Inicial:**

Solamente individuos factibles sobre la frontera deberían ser considerados para conformar la población inicial. Por lo tanto, es necesario implementar un procedimiento *ad-hoc* para generar tales soluciones sobre la frontera. Este procedimiento podría ser implementado como un proceso aleatorio o greedy. Sin embargo, las diferentes alternativas tienen que ser consideradas según el problema particular

a resolver. Esta puede ser considerada una desventaja del enfoque. Sin embargo, la generación de soluciones en general puede hacerse incorporando paso a paso una componente hasta que alguna de las restricciones del problema sea violada (MKP, MISP) o hasta que todas las restricciones sean satisfechas (SCP). Es decir, el proceso es uniforme y lo que varía son las condiciones para seguir agregando componentes según el problema en cuestión y las respectivas restricciones.

■ Operadores genéticos:

Cualquiera de los dos, el crossover (χ) o la mutación (ψ), deben ser diseñados de manera tal que el conjunto de puntos que forman la frontera ($\mathcal{F}_b \subseteq \mathcal{F}$) sea cerrado bajo la aplicación de cualquiera de estas operadores, es decir, $\chi : \mathcal{F}_b \times \mathcal{F}_b \rightarrow \mathcal{F}_b$, y $\psi : \mathcal{F}_b \rightarrow \mathcal{F}_b$. Aquí se asume que el crossover toma como argumentos 2 soluciones padre y da como resultado un hijo, mientras que la mutación toma como argumento un padre y genera un hijo.

Crossover

El procedimiento general para aplicar el operador de crossover de frontera está dividido en dos etapas (figura 5.9). El algoritmo en la figura 5.10 muestra la estructura global de la respectiva implementación del crossover. Dicha figura muestra dos posibles situaciones para un par de puntos según las restricciones del problema considerado. Sobre la izquierda los puntos son representativos de una situación para problemas con restricciones del tipo $g_j(\mathbf{x}) \leq b_j$ y sobre la derecha, lo son para restricciones del tipo $g_j(\mathbf{x}) \geq b_j$. En la primer etapa, para cualquiera de los dos casos, el hijo \mathbf{z} (solución parcial) hereda la información común compartida por ambos padres. Este punto \mathbf{z} puede ser una solución factible o no factible³,

³Se consideran solamente las componentes $z_i = 1$.

pero esto depende del problema bajo consideración como se verá en los próximos capítulos. Por ejemplo, cuando todas las restricciones son del tipo $g_j(\mathbf{x}) \leq b_j$, la solución \mathbf{z} será factible dado que \mathbf{x} e \mathbf{y} son factibles y además \mathbf{z} hereda solamente las componentes de la solución que hacen factible a los padres. Sin embargo, cuando las restricciones son del tipo $g_j(\mathbf{x}) \geq b_j$, el hijo \mathbf{z} probablemente será no factible debido a la supresión implícita de algunas componentes de \mathbf{x} e \mathbf{y} respectivamente. Más aun, considerando que dichas soluciones están sobre la frontera, la eliminación eventual de cualquier componente puede tornarla no factible. Luego, con una alta probabilidad, \mathbf{z} no va a pertenecer a la frontera dado que ya sea, 1) más componentes del problema pueden ser añadidas a la solución sin violar ninguna restricción del problema ($g_j(\mathbf{x}) \leq b_j$) o bien, 2) más componentes del problema deben ser añadidas para satisfacer las restricciones del problema ($g_j(\mathbf{x}) \geq b_j$).

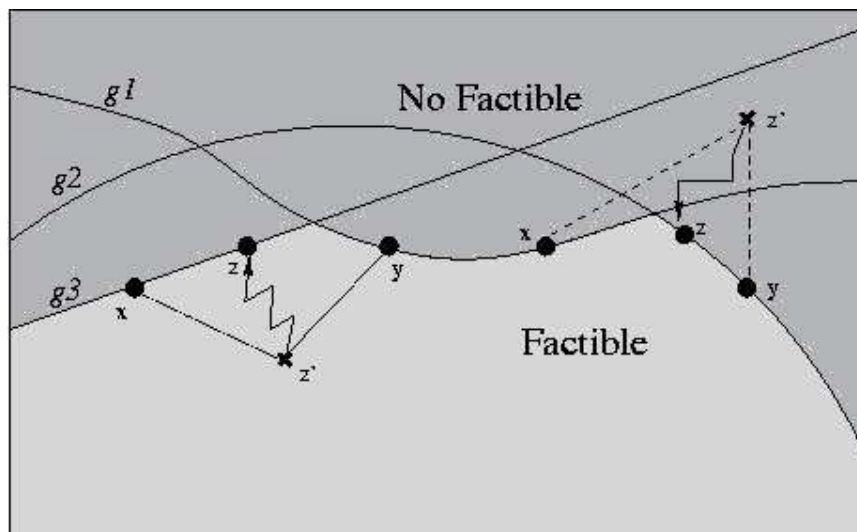


Figura 5.9: Visualización del operador de crossover de frontera

Durante la fase 2, la solución \mathbf{z} es completada como una solución (de frontera) a través de un proceso iterativo (ver figura 5.10, línea 9).

La selección de una nueva componente, representada por la fun-

```

1. bound_crossover ( x,y, z: vector binario)
2. {
3.
4.   // Etapa 1
5.   z[i] = • for i ∈ {1, ...n}
6.   for i ∈ {1, ..., }
7.     if ( x[i] == y[i] ) z[i] = x[i] = y[i]
8.
9.   // Etapa 2
10.  while ( S ≠ ∅ )
11.  {
12.    j = seleccionar(S)
13.    actualizar(S, j)
14.    c[j] = 1
15.  }
16.  z[i] = 0 for i = 1, .., n and z[i] = •
17.  return z
18. }
```

Figura 5.10: Estructura general del operador de crossover de frontera.

ción *seleccionar*, puede ser implementada de maneras diferentes. Por ejemplo, ésta podría ser una selección aleatoria de las componentes faltantes, una selección greedy basada en el conocimiento del problema, o un enfoque intermedio. La función *actualizar* está a cargo de la eliminación de la componente *j* del conjunto *S* y de todas aquellas componentes que se tornan no factibles por la incorporación de *j* a la solución **z**. Sin embargo existen otras consideraciones a tener en cuenta según el problema que se pretende resolver (ver capítulo 5).

Mutación

Este operador también puede ser implementado en dos etapas (figura 5.11). En la primera etapa (desde la línea 7) la solución es

modificada cambiando valores en ciertas posiciones de 1 a 0, es decir que algunas componentes (podrían ser una o más) son eliminadas de la solución. La condición *condicion_para_eliminar* se plantea de manera tal que unos pocas componentes son eliminadas del padre \mathbf{x} . La selección de la componente (función *seleccionar_para_eliminar*) a ser eliminada está basada también en valores heurísticos y es implementada de manera estocástica. En la segunda etapa (desde la línea 16) toma lugar un proceso inverso para la incorporación de componentes a la solución, es decir, algunas posiciones son cambiadas de 0 a 1. El proceso de chequeo *check*(S, \mathbf{x}) en la línea 17 determina si la solución \mathbf{x} puede ser completada como una solución de frontera usando algunos elementos del conjunto S . Si esta condición no es satisfecha, la solución original es restablecida (línea 25) y retornada sin sufrir ningún cambio.

5.6. Enfoque de frontera en ACO

Caso continuo:

El AH para resolver problemas de optimización numéricos con restricciones está basado en trabajos de Bilchev et al. [22, 24, 21]. Tal como se mencionó previamente, las primeras aplicaciones del enfoque ACO en casos continuos estaba esencialmente orientado a problemas sin restricciones. En este sentido, solamente unos pocos experimentos han sido reportados en relación a su aplicación a problemas numéricos con restricciones a través del uso de funciones de penalización, aunque ninguno de dichos experimentos incluye operadores de frontera. De manera similar al enfoque evolutivo, cada solución es representada como un vector de punto flotante. Sin embargo, el enfoque ACO no incorpora el concepto de recombinación o crossover. En cambio, implementa una especie de operador de mutación a fin de explorar el espacio de búsqueda. De esta manera, este operador debe ser diseñado en forma tal que solamente sean

```

1. bound_mutation ( x,z: vector binario)
2. //  $C = \{i/x[i] = 1\}$  es el conjunto de componentes candidatas
   para ser eliminadas de  $x$ 
3. //  $D = \emptyset$  es el conjunto de componentes eliminados de x.
4. //  $S = \{i/x[i] = 0\}$  es el conjunto de posibles candidatos
   a ser incorporados en la solución  $x$ .
5.
6. {
7. // Etapa 1
8.   while ( condicion_para_eliminar )
9.   {
10.     $j = \text{seleccionar\_para\_eliminar}(C)$ 
11.     $C = C - \{j\}$ 
12.     $D = D \cup \{j\}$ 
13.     $x_j = 0$ 
14.  }
15.
16. // Etapa 2
17. if ( chequear(  $S$ , x ) )
18.   while ( $S \neq \emptyset$  )
19.   {
20.     $j = \text{seleccionar}(S)$  /* igual al crossover (Etapa 2) */
21.    actualizar( $S, j$ )
22.     $x_j = 1$ 
23.   }
24. else
25.    $x_i = 1$ , for  $i \in D$ 
26.   return x
27. }
```

Figura 5.11: Estructura general de la mutación de frontera.

generadas soluciones factibles de frontera (por ejemplo, los operadores de mutación definidos para algoritmos evolutivos pueden ser usados directamente en este contexto). Otras componentes del algoritmo variarán

también según el problema bajo consideración. De la misma manera, los mismos métodos de penalización que se definieron para los algoritmos evolutivos, también podrían ser usados aquí sin ningún problema.

Con el objeto de aplicar el enfoque ACO en un contexto de problemas continuos, la siguiente estrategia es considerada: se plantea una estructura discreta para modelar un nido o población de puntos dispersos en el espacio (continuo) de búsqueda del problema. La estrategia determina un conjunto de direcciones $\{d_1, d_2, \dots, d_k\}$, donde k es un parámetro del método ($k = 4$ en la figura 5.12). Cada dirección d_i es representada como un vector (o punto) en un espacio de búsqueda n -dimensional.

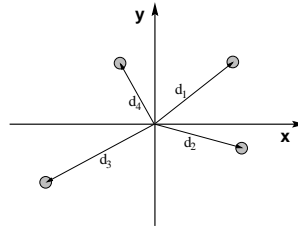


Figura 5.12: Nido con cuatro puntos

La estructura general del algoritmo ACO según [22] es mostrado en la figura 5.13. Antes de continuar es importante destacar que la propuesta inicial de Bilchev [22] asume que la aplicación del algoritmo ACO corresponde a una etapa que es continuación de la aplicación de un enfoque evolutivo o similar el cual se supone convergió a una determinada región del espacio de búsqueda y a partir de este punto el enfoque ACO mejoraría los resultados obtenidos de la explotación de las soluciones encontradas hasta el momento. Esto implica, según la visión original, que este método se podría aplicar preferentemente en situaciones en las que previamente se ha arribado a buenas soluciones y que las mismas podrían ser mejoradas. Si bien esta visión del problema puede ser válida, no existe nada que impida aplicar el método directamente partiendo desde un conjunto de soluciones generadas según un determinado criterio. Nuestra propuesta para la aplicación el método en cuestión parte

de un conjunto de puntos generados en forma aleatoria con valores uniformemente distribuidos los que representan las direcciones de búsqueda inicial para el algoritmo.

Adicionalmente se establece un valor R_i (para $1 \leq i \leq n$) definido a fin de establecer un radio de búsqueda por cada dimensión de los puntos del nido inicial. El valor de R_i debe estar sujeto a los valores límites de la i -ésima variable del problema. Dicho radio de búsqueda determina la máxima distancia en cada dimensión en las que el algoritmo podrá realizar la búsqueda en cada *ciclo* de su ejecución.

```

procedure Algoritmo ACO
begin
    t = 0
    inicializar  $A(t)$ 
    evaluar  $A(t)$ 
    while (not condicion_de_terminacion ) do
        begin
            t = t + 1
            actualizar_rastro  $A(t)$ 
            ubicar_hormigas  $A(t)$ 
            evaluar  $A(t)$ 
            evaporar  $A(t)$ 
        end
    end

```

Figura 5.13: Estructura general de un algoritmo ACO para problemas continuos.

De esta manera, **inicializar** $A(t)$ “distribuye” las hormigas en varias direcciones (cada hormiga genera una solución y en general existirán más hormigas que direcciones); **evaluar** $A(t)$ es una llamada a la función objetivo para todas las hormigas (evaluación de todos los puntos); **actualizar_rastro** $A(t)$ está a cargo de añadir el rastro respectivo sobre cada dirección en forma proporcional al fitness de cada una de las hormigas que eligieron dicha dirección. Luego, **ubicar_hormigas** $A(t)$ redistribuye las hormigas seleccionando las respectivas direcciones según

una *Ruleta* basada en los valores proporcionales a la cantidad de rastro sobre cada una de las direcciones en relación al rastro total acumulado en todas las direcciones. Una vez escogida una dirección de búsqueda, el algoritmo realiza una búsqueda en un paso aleatorio desde el punto de mejor valor objetivo ubicado en la dirección respectiva, es decir, todas las hormigas sobre esa dirección parten del mismo punto para explorar el vecindario del mismo. Finalmente, **evaporar** $A(t)$ reduce la cantidad de rastro acumulado en cada dirección. El paso aleatorio sobre cada dimensión del problema que permite explorar el vecindario de un determinado punto puede ser implementado como:

$$\Delta_{R_i}(t) = R_i \cdot (1 - r^{(1-t/T)^b}) \quad (5.2)$$

donde R_i es un parámetro fijo dependiente del problema; r es un número aleatorio de intervalo $[0.,1]$; T es el número máximo de ciclos o iteraciones del algoritmo y b es un parámetro adicional que determina el grado de no-uniformidad. En consecuencia, $\Delta_{R_i}(t)$ retorna un valor en el rango $[0..R_i]$ tal que la probabilidad de $\Delta_{R_i}(t)$ tienda a 0 se incrementa a medida que t (ciclo corriente) se incrementa.

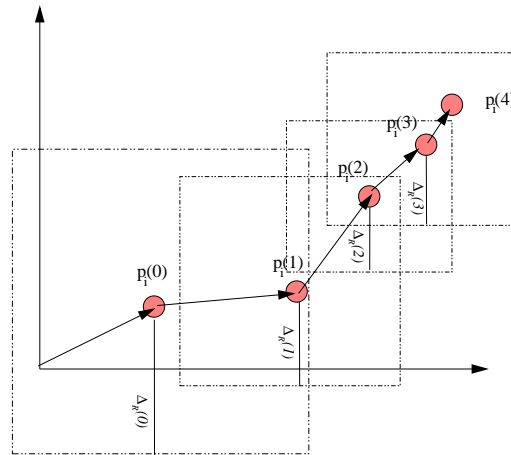


Figura 5.14: Puntos generados en varios pasos respetando su vecindario respectivo sobre cada dimensión.

La figura 5.14 representa la evolución de un vector o punto bidimensional $p_i(t)$ sobre la dirección i para $t \in \{0, 1, 2, 3, 4\}$. En este caso y

para simplificar la explicación, asumimos que $R_1 = R_2 = R$. Luego, un rectángulo para nuestro ejemplo representa el vecindario de un punto particular. Siguiendo nuestro algoritmo, en el primer ciclo un número fijo de hormigas son distribuidas en las k direcciones—es decir, las hormigas que fueron asignadas sobre la dirección i en el tiempo $t = 0$ comenzarán la búsqueda desde el vector $p_i(0)$. Por ejemplo, en la figura 5.14, $p_i(0)$ es el vector sobre la dirección i y $p_i(1)$ es el mejor punto encontrado por las hormigas asignadas a la dirección i usando un radio de búsqueda $\Delta_R(0)$ ⁴. A medida que t se incrementa, nuevas regiones del espacio de búsqueda son exploradas en cada una de las direcciones y en particular en la dirección i a través de $p_i(2)$, $p_i(3)$, $p_i(4)$ y así sucesivamente. Es importante resaltar la similitud entre los saltos que este algoritmo realiza durante el proceso de exploración del espacio y la forma en que es realizado por ejemplo, por Simulated Annealing. En el caso del algoritmo ACO, la estrategia de búsqueda podría ser considerada como un método de trayectoria combinado con una "simultaneidad" en la exploración por varias "direcciones." a la vez, enfatizando o memorizando a través del rastro, aquellas que mostraron un mejor desempeño.

Es claro que aquellas direcciones sobre las cuales los puntos encontrados no producen mejoras en las mejores soluciones encontradas sobre cada dirección, ocurrirá que las mismas no participarán en el proceso de acumulación de rastro dada la baja calidad de las soluciones encontradas y por consiguiente el proceso inverso de evaporación diversificará la atención de las hormigas a otras direcciones más prometedoras. A fin de evitar una convergencia acelerada del método hacia determinadas direcciones en desmedro e otras potencialmente útiles, es posible aquí también establecer un límite sobre los valores mínimos y máximos del rastro⁵ La disminución de la cantidad de rastro sobre ciertas direcciones puede ser visto como el proceso de *food source exahusting* o agotamiento de la

⁴Debería notarse que $\Delta_R(t)$ no es un valor monótono decreciente

⁵Siguiendo la propuesta de la variante SH-MM descrita en sección 3.1.3

fuentes de alimentos.

La principal característica de este método es que comprende dos niveles de abstracción,

1. *búsqueda individual*: describe la estrategia de búsqueda individual de cada hormiga—es decir, el método empleado para explorar el vecindario de un punto determinado. Dicho método podría ser hill-climbing, el descenso del gradiente, etc.
2. *cooperación*: involucra el intercambio de información entre los agentes el cual tiene por objetivo principal realizar un esfuerzo conjunto dirigido a ciertas direcciones en el espacio de búsqueda. Esta información está representada por τ , donde τ_i es la acumulación de rastro sobre la dirección i . Durante la distribución sobre las direcciones y antes de realizar la búsqueda en el vecindario de un punto específico, las hormigas seleccionan la dirección en forma probabilística según la fórmula:

$$P_i(t) = \frac{[\tau_i(t)]^\alpha \cdot [\eta_i(t)]^\beta}{\sum_k [\tau_k(t)]^\alpha \cdot [\eta_k(t)]^\beta} \quad (5.3)$$

en la cual η_i es uno (tal cual ha sido considerada a lo largo de los experimentos en esta tesis o podría incorporar información del problema⁶ y α y β definen al igual que en la formulación original de este enfoque, la importancia relativa del rastro o dinámica de la colonia de hormigas y conocimiento del problema en la conformación de los respectivos valores de probabilidad.

En cuanto a la variación de la cantidad de rastro en cada dirección (τ_i), el algoritmo ACO modifica τ_i según (1) $\tau_i = \tau_i + \Delta\tau_i$ en `update_trail A(t)` y (2) $\tau_i = \rho \cdot \tau_i$ en `evaporate_trail A(t)`,

⁶Por ejemplo, para algunos dominios y problemas, se podría usar en caso de estar disponible, información del gradiente o similar.

donde $\Delta\tau_i$ es una cantidad proporcional al fitness de la mejor hormiga (mejor solución) encontrada en la dirección i y $(1 - \rho)$ es el respectivo coeficiente de evaporación.

Caso discreto:

Como fue descripto en el sección 3.1, los AHs puede ser aplicado directamente a problemas de optimización combinatoria que puedan ser caracterizados como un grafo $G = (C, L)$, donde C es un conjunto finito de *componentes* y $L \subseteq C \times C$ el conjunto de *conexiones* entre dichas componentes. Las soluciones para el problema de optimización pueden en consecuencia, ser expresadas en términos de pasos factibles sobre el grafo G . Luego, los algoritmos ACO pueden ser usados para encontrar pasos factibles de costo mínimo (secuencias) con respecto a un número de restricciones Ω ⁷. Sin embargo, esta formulación podría implicar que el enfoque ACO no es adecuado para ser aplicado a la resolución de problemas de subconjunto dado que en este tipo de problemas eventuales conexiones entre las componentes del problemas son irrelevantes y el método, en su concepción original, asume que el problema a resolver admite un grafo como representación de manera tal que las hormigas puedan “caminar” a través del grafo con el objeto de conectar los nodos y así construir una solución que por lo general conforma una permutación de enteros como lo es el caso del TSP, problemas de scheduling, etc. Sin embargo, la idea original propuesta por Dorigo [39] en cuanto al uso del rastro de feromona sobre las conexiones del grafo como una forma de reforzar ciertos pasos del grafo puede ser llevada en forma sencilla y directa a problemas en los cuales las conexiones (ordenamiento entre las componentes del problema) no sean relevantes para expresar una posible solución. Esto implica la flexibilización de ciertos requerimientos para incluir otro tipo de problemas y al mismo tiempo manteniendo la

⁷Por ejemplo, en el problema de viajante de comercio definido en la sección 3.1, C es el conjunto de ciudades, L es el conjunto de arcos que conectan las ciudades, y Ω indica que una solución particular debe ser un circuito Hamiltoniano.

idea original en cuanto a la característica cooperativa del enfoque.

En lo que sigue, se presenta una nueva versión del Sistema de Hormigas presentado en el sección 3.1. La nueva versión del SH propuesto está diseñado para ser aplicado a problemas de subconjunto. Cada solución para un problema de subconjunto puede ser visualizada como un conjunto de componentes tomadas del conjunto total de las componentes del problema. Luego, mientras más restringido sea el problema, más complejo será el proceso de seleccionar las componentes que serán parte de la solución final.

El SH para problemas cuya solución es expresada a través de una permutación (SH-P) y aquel donde la solución es expresada como un subconjunto (SH-S) tienen muchas características en común. Sin embargo, en un SH-P el rastro de feromona es depositado sobre los pasos (conexiones entre las componentes del problemas) mientras que en un SH-S no existen pasos que conecten items o componentes del problema para conformar una posible solución⁸. Un SH-S toma ventaja de una de las ideas centrales involucradas en el proceso de selección de un SH-P: “mientras mayor sea la cantidad de feromona depositada sobre un paso particular, mayor es la probabilidad de seguir dicho paso”. Esta idea ha sido adaptada aquí de la siguiente manera: “Mientras mayor sea la cantidad de feromona depositada sobre una componente particular, mayor será la probabilidad de elegir dicha componente como parte de la solución”. En otras palabras, el principio de acumulación de feromona sobre *pasos* es cambiado a las *componentes* del problema. De la misma manera que en el SH original, es posible usar una heurística local para añadir información del problema en calculo de probabilidades (ver Algoritmo 3.3, línea 9) para la selección de la próxima componente. La diferencia es que dichos valores heurísticos están consecuentemente asociados a las componentes en vez de las conexiones entre ellas, es decir, η_{ij} es cambiado por η_i .

⁸A lo largo de esta tesis, el término *item* es también usado como componente.

Es interesante explorar las similitudes y diferencias entre este tipo de SHs a fin de clarificar en cómo el enfoque es adaptado para poder ser aplicado a otro tipo de problemas de optimización combinatoria: por ejemplo, MKP, SCP o MISIP.

Del conjunto C de componentes ($|C| = n$ el tamaño del problema) se debe elegir el mejor subconjunto que maximice (minimice) una función objetivo y que satisfaga ciertas restricciones del problema. Dado que no existe el concepto de paso, las componentes son lo único relevante para el algoritmo. Por lo tanto, es necesario clarificar cómo aplicar los conceptos descritos en secciones previas para problemas que involucraban componentes y conexiones entre ellas. La diferencia principal radica en lo siguiente. En problemas cuya solución se pueda expresar como una permutación, la secuencia $\tilde{s}^k = \langle i_1, i_2, \dots, i_j \rangle^9$ y el conjunto $R^k = \{i_{j+1}, i_{j+2}, \dots, i_n\}^{10}$ representan respectivamente una solución parcial al problema y el conjunto de componentes remanentes a ser consideradas para completar la solución el ordenamiento de los n items del conjunto C . El proceso de selección de la próxima componente del conjunto R involucra probabilidades $P_{i_j i_p}^k(t)$ ($p \in \{j+1, j+2, \dots, n\}$), las cuales dependen del rastro $\tau_{i_j i_p}$ sobre el arco (i_j, i_p) y la heurística local $\eta_{i_j i_p}$ (Figura 5.15).

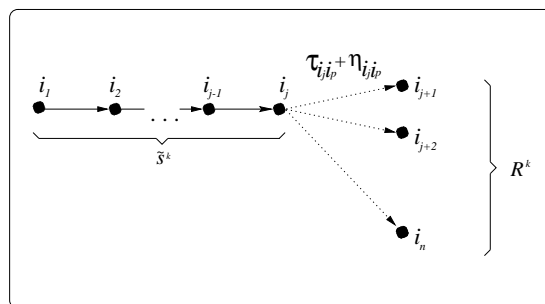


Figura 5.15: Una secuencia que representa una solución parcial \tilde{s}^k en el paso j durante un ciclo particular.

Por otro lado, en problemas de subconjunto, el interés no está pues-

⁹La letra minúscula 's' representa una secuencia (ver sección 3.1).

¹⁰ $R^k = C - \mathcal{M}^k$ según la descripción de la metaheurística ACO presentada en sección 3.1.

to en un posible orden de las componentes (Por ejemplo, un tour para TSP). Por lo tanto, una solución parcial es representada como un conjunto $\tilde{S}^k = \{i_1, i_2, \dots, i_j\}$ para la cual se asume que el elemento más reciente incorporado a la solución, i_j , no influye en el proceso de selección de la próxima componente, sino que sólo se consideran los elementos remanentes factibles (Figura 5.16). En este caso, el conjunto R^k es renombrado como A^k y el índice de la última componente i_n es cambiado por i_{n_a} ($n_a \leq n$). Este cambio en la notación será explicado más adelante en esta sección.

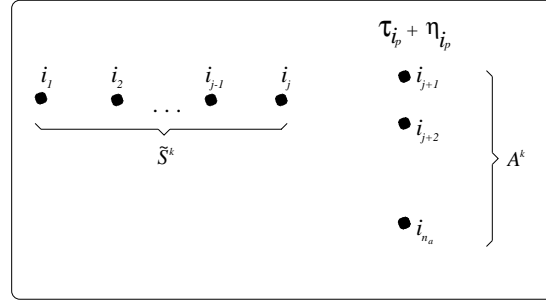


Figura 5.16: Un conjunto que representa una solución parcial \tilde{S}^k en el paso j durante un ciclo particular.

Esta modificación implica que el correspondiente rastro de feromona es depositado en cada elemento del conjunto C , con el significado que los elementos con un mayor nivel de feromona son mejores candidatos para su selección. De esta manera, la intensidad del rastro de feromona sobre la componente i en el tiempo $t + 1$ está dado por:

$$\tau_i(t + 1) = (1 - \rho)\tau_i(t) + \Delta\tau_i(t), \quad (5.4)$$

donde $\Delta\tau_i(t) = \sum_{k=1}^a \Delta\tau_i^k(t)$, es decir, la cantidad acumulada de la contribución de cada agente $\Delta\tau_i^k(t)$ en el sistema (La ecuación 5.4 combina las ecuaciones 3.1 y 3.2 dada para TSP). En otras palabras, esa contribución es la cantidad de rastro de feromona depositado en la componente i por la k -ésima hormiga en el tiempo t . Esta cantidad es obte-

nida según la siguiente fórmula:

$$\Delta\tau_i^k(t) = \begin{cases} G(L_k), & \text{si } k\text{-ésima hormiga incorpora la componente } i \\ 0, & \text{en otro caso.} \end{cases} \quad (5.5)$$

En la ecuación 5.5 la función G depende del problema y arroja la cantidad de rastro añadido a la componente i . Usualmente, $G(L_k) = Q/L_k$ o $G(L_k) = QL_k$, para problemas de maximización y minimización respectivamente (Q es una constante que depende del problema). L_k es el valor objetivo de la solución obtenida por el hormiga k . Asimismo, la heurística local debería asignar un valor a cada elemento sin considerar ningún tipo de conexión entre ellas dado que en este contexto no son parte del problema. Para los problemas de subconjunto considerados en esta tesis, la versión del Sistema de Hormigas usa un tipo especial de heurística que toma en cuenta dos cosas, información del problema y la solución parcial en proceso de construcción por una hormiga en particular.

Luego, para una solución parcial $\tilde{S}^k(t) = \{i_1, \dots, i_j\}$ que está siendo construida por la hormiga k , la probabilidad $P_{i_p}^k(t)$ de seleccionar la componente i_p ($p \in \{j+1, j+2, \dots, n_a\}$) como la próxima a incorporar en la solución viene dada por

$$P_{i_p}^k(t) = \frac{a_{i_p}}{\sum_{l \in A^k(t)} a_{i_l}} \quad (5.6)$$

donde $A^k(t) = \{i_{j+1}, \dots, i_{n_a}\} \subseteq C - \tilde{S}^k(t)$ es el conjunto de componentes factibles remanentes. De manera similar, el conjunto de componentes candidatas representadas por \mathcal{N}_i^k en la formulación original, no es considerado en la nueva versión. En su lugar es definido el conjunto A^k en la ecuación Eq. 5.6. En consecuencia, la tabla de ruteo respectiva $\mathcal{A}_i^k = [a_{ij}]$ es renombrada como “tabla de beneficios” de la siguiente manera: $\mathcal{A} = [a_j]$, con $a_j = \tau_j(t)^\alpha \eta_j(\tilde{S}^k(t))^\beta$ donde $\tau_i(t)$ es la cantidad de feromona sobre la componente i , y $\eta_{i_p}(\tilde{S}^k(t))$ representa el valor heurístico de la componente i de acuerdo a la solución parcial en proceso

de construcción por la hormiga k . Así, mientras mayor sea el valor de τ_{i_p} y/o $\eta_{i_p}(\tilde{S}^k(t))$, se espera un mayor beneficio en la calidad de la solución por la inclusión de la componente i_p . La figura 5.17 muestra una versión general del SH para problemas de subconjunto basado en el algoritmo descrito en la figura 3.3 para TSP.

```

1. SH()
2. {
3.   inicializar();
4.   while (criterio_de_terminación != false)
5.     for  $k = 1$  to  $n'$  do
6.        $j = \text{get\_start\_item}()$ ;  $S^k = \{j\}$ ;
7.        $A^k = C - \{j\} - U$ ;
8.       while( $A^k \neq \emptyset$ )
9.         Seleccionar componente  $j$  a ser incorporada
           con probabilidad  $P_j^k = \frac{a_j}{\sum_{l \in A^k} a_l}$ 
10.         $S^k = S^k + \{j\}$ 
11.         $A^k = A^k - \{j\} - U$ 
12.      end
13.      calcular  $J_\psi$  el valor objetivo de la solución generada  $\psi^k$ 
14.      guardar la mejor solución hasta el momento
15.    end
16.    actualizar los niveles de rastro  $\tau_i$  sobre todas
           las componentes (Eq. 5.4)
17. end
18. Imprimir la mejor solución encontrada
19. }
```

Figura 5.17: El Sistema de Hormigas general para problemas de subconjunto.

La condición ($A^k = \emptyset$) implica que ninguna otra componente puede ser añadida a la solución, es decir, que la solución está en la frontera. Se puede observar en las líneas 7 y 11 que A^k es modificado a través de la eliminación de la componente j recientemente seleccionada y del conjunto de componentes U , el cual representa el conjunto de componen-

tes no factibles debido a la inclusión de j en la solución¹¹. La condición para calificar como componente no factible, depende del problema. Por ejemplo para el problema del Conjunto Máximo Independiente (definido más adelante en esta sección), el conjunto U representa algunas (probablemente todas) de las componentes que están conectadas al vértice j .

5.7. Problemas estudiados

En esta sección se describen los problemas seleccionados para la aplicación y análisis del comportamiento de las metaheurísticas. La descripción incluye un conjunto importante de problemas bien conocidos de optimización combinatoria y numérica. Todos estos problemas han sido estudiados a través de la aplicación de diferentes enfoques, desde métodos tradicionales a las modernas metaheurísticas actuales.

5.7.1. Variables Continuas

■ G01

Problema propuesto por [58] y consiste en maximizar la función:

$$f(\mathbf{x}) = 5 \cdot \sum_{i=1}^4 x_i - 5 \cdot \sum_{i=1}^4 x_i^2 - \sum i = 513x_i \quad (5.7)$$

¹¹Esta es la razón por la que $A^k \subseteq C - \tilde{S}$ y por ende, $n_a \leq n$. De cualquier manera la igualdad no se cumple para los problemas combinatorios tratados en esta tesis.

sujeto a:

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

$$g_2(\mathbf{x}) = 2x_1 + 2x_3 + x_{10} + x_{12} - 10 \leq 0$$

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

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

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

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

$$g_7(\mathbf{x}) = -2x_4 - x_5 + x_{10} \leq 0$$

$$g_8(\mathbf{x}) = -2x_6 - x_7 + x_{11} \leq 0$$

$$g_9(\mathbf{x}) = -2x_8 - x_9 + x_{12} \leq 0$$

donde $0 \leq x_i \leq 1 (i = 1, \dots, 9)$, $0 \leq x_i \leq 100 (i = 10, 11, 12)$ y $0 \leq x_{13} \leq 1$. El óptimo global es en el punto $\mathbf{x} = (1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 1)$, en el cual seis restricciones están activas: g_1, g_2, g_3, g_7, g_8 , y g_9 y el valor de la función objetivo es $f(\mathbf{x}) = -15$.

■ G02: Función de Keane

Este es un interesante de problema de optimización numérica restringido el cual fue propuesto por Keane [96] y consiste en maximizar la función:

$$F_K(\mathbf{x}) = \left| \frac{\sum_{i=1}^n \cos^4(x_i) - 2 \prod_{i=1}^n \cos^2(x_i)}{\sqrt{\sum_{i=1}^n i x_i^2}} \right|, \quad (5.8)$$

sujeto a:

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

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

donde $0 \leq x_i \leq 10$ para $1 \leq i \leq n$.

La función F_K (ver figura 5.18) es no lineal y su máximo global es desconocido, aunque se supone que estaría en algún punto cercano al origen de coordenadas. Este problema tiene una restricción no lineal y una lineal, la última, es no activa alrededor del origen, lo

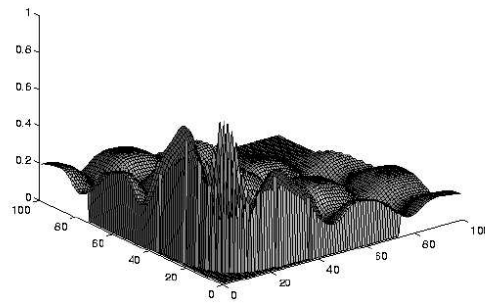


Figura 5.18: Función de Keane para dos variables.

que da indicios suficientes para la aplicación del enfoque de frontera. El máximo valor de la función objetivo depende del valor de n dado que esta es una función escalable. El valor que ha sido usado principalmente para estudiar este problema es $n = 20$ para el cual el mejor valor conocido de la función objetivo es 0,803619.

■ G03

Este problema fue propuesto por Michalewicz et al. [120] y consiste en la maximización de la siguiente función:

$$F_P(\mathbf{x}) = (\sqrt{n})^n \cdot \prod_{i=1}^n x_i, \quad (5.9)$$

sujeto a:

$$\sum_{i=1}^n x_i^2 = 1$$

donde $0 \leq x_i \leq 1 (i = 1, \dots, n)$. El óptimo global para $n = 10$ es $\mathbf{x} = (1/\sqrt{n}, \dots, 1/\sqrt{n})$ y $f(\mathbf{x}) = 1$.

■ G04

Problema propuesto en [86] y consiste en minimizar la siguiente función:

$$f(\mathbf{x}) = 5,3578547x_3^2 + 0,8356891x_1x_5 + 37,293239x_1 - 40792,141 \quad (5.10)$$

sujeto a:

$$\begin{aligned} g_1(\mathbf{x}) &= 85,334407 + 0,0056858x_2x_5 + 0,0006262x_1x_4 - 0,0022053x_3x_5 - 92 \leq 0 \\ g_2(\mathbf{x}) &= -85,334407 - 0,0056858x_2x_5 - 0,0006262x_1x_4 + 0,0022053x_3x_5 \leq 0 \\ g_3(\mathbf{x}) &= 80,51249 + 0,0071317x_2x_5 - 0,0029955x_1x_2 + 0,0021813x_3^2 - 100 \leq 0 \\ g_4(\mathbf{x}) &= -80,51249 - 0,0071317x_2x_5 + 0,0029955x_1x_2 - 0,0021813x_3^2 \leq 0 \\ g_5(\mathbf{x}) &= 9,300961 + 0,0047026x_3x_5 + 0,0012547x_1x_3 + 0,0019085x_3x_4 - 25 \leq 0 \\ g_6(\mathbf{x}) &= -9,300961 - 0,0047026x_3x_5 - 0,0012547x_1x_3 - 0,0019085x_3x_4 \leq 0 \end{aligned}$$

donde $78 \leq x_1 \leq 102$, $33 \leq x_2 \leq 45$ y $27 \leq x_i \leq 45 (i = 3, 4, 5)$. La solución óptima es $\mathbf{x} = (78, 33, 29,995256025682, 45, 36,775812905788)$ y $f(\mathbf{x}) = -30665,539$. Las restricciones activas en este punto son g_1 y g_6 .

■ G05

Problema propuesto en [169] y consiste en la minimización de la siguiente función:

$$f(\mathbf{x}) = 3x_1 + 0,000001x_1^2 + 2x_2 + (0,000002/3)x_2^3 \quad (5.11)$$

sujeto a:

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

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

$$h_3(\mathbf{x}) = 1000\text{sen}(x_3 - 0,25) + 1000\text{sen}(-x_4 - 0,25) + 894,8 - x_1 = 0$$

$$h_4(\mathbf{x}) = 1000\text{sen}(x_3 - 0,25) + 1000\text{sen}(x_3 - x_4 - 0,25) + 894,8 - x_2 = 0$$

$$h_5(\mathbf{x}) = 1000\text{sen}(x_4 - 0,25) + 1000\text{sen}(x_4 - x_3 - 0,25) + 294,8 = 0$$

donde $0 \leq x_1 \leq 1200$, $0 \leq x_2 \leq 1200$, $-0,55 \leq x_3 \leq 0,55$, y $-0,55 \leq x_4 \leq 0,55$. La mejor solución conocida [104] es $\mathbf{x} = (679,94453, 1026,067, 0,1188764, -0,3962336)$ y $f(\mathbf{x}) = 5126,4981$.

■ G06

Este problema fue propuesto por Floudas et al. [58] (Figura 5.19) y consiste en maximizar la siguiente función:

$$f(\mathbf{x}) = (x_1 - 10)^3 + (x_2 - 20)^3, \quad (5.12)$$

sujeto a:

$$g_1(\mathbf{x}) = (x_1 - 5)^2 + (x_2 - 5)^2 - 100 \geq 0,$$

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

donde $13 \leq x_1 \leq 100$ y $0 \leq x_2 \leq 100$. La solución global conocida es $\mathbf{x}^* = (14,095, 0,84296)$, y $f(\mathbf{x}^*) = -6961,81381$. En suma, es importante notar que ambas restricciones son activas en el mejor punto conocido u óptimo.

■ G07

Problema propuesto en [169] y consiste en la minimización de la siguiente función:

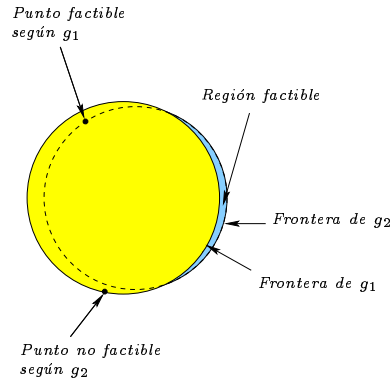


Figura 5.19: Problema G6. Función de Floudas-Pardalos.

$$\begin{aligned}
 f(\mathbf{x}) = & x_1^2 + x_2^2 + x_1x_2 - 14x_1 - 16x_2 + (x_3 - 10)^2 \\
 & + 4(x_4 - 5)^2 + (x_5 - 3)^2 \\
 & + 29x_6 - 1)^2 + 5x_7^2 + 7(x_8 - 11)^2 + 2(x_9 - 10)^2 \\
 & + (x_{10} - 7)^2 + 45
 \end{aligned} \quad (5.13)$$

sujeto a:

$$\begin{aligned}
 g_1(\mathbf{x}) &= -105 + 4x_1 + 5x_2 - 3x_7 - 9x_8 \leq 0 \\
 g_2(\mathbf{x}) &= 10x_1 - 8x_2 - 17x_7 + 2x_8 \leq 0 \\
 g_3(\mathbf{x}) &= -8x_1 + 2x_2 + 5x_9 - 2x_{10} - 12 \leq 0 \\
 g_4(\mathbf{x}) &= 3(x_1 - 2)^2 + 4(x_2 - 3)^2 + 2x_3^2 - 7x_4 - 120 \leq 0 \\
 g_5(\mathbf{x}) &= 5x_1^2 + 8x_2 + (x_3 - 6)^2 - 2x_4 - 40 \leq 0 \\
 g_6(\mathbf{x}) &= x_1^2 + 2(x_2 - 2)^2 - 2x_1x_2 + 14x_5 - x_6 \leq 0 \\
 g_7(\mathbf{x}) &= 0,5(x_1 - 8)^2 + 2(x_2 - 4)^2 + 3x_5^2 - x_6 - 30 \leq 0 \\
 g_8(\mathbf{x}) &= -3x_1 + 6x_2 + 12(x_9 - 8)^2 - 7x_{10} \leq 0
 \end{aligned}$$

donde $-10 \leq x_i \leq 10 (i = 1, \dots, 10)$. La solución óptima es $\mathbf{x} = (2,171996, 2,363683, 8,773926, 5,095984, 0,9906548, 1,430574, 1,3221644, 9,828726, 8,280092, 8,375927)$ para el cual $f(\mathbf{x}) = 24,306209$. En este punto seis restricciones están activas: g_1, g_2, g_3, g_4, g_5 , y g_6 .

■ G09

Problema propuesto en [169] y consiste en la minimización de la siguiente función:

$$f(\mathbf{x}) = (x_1 - 10)^2 + 5(x_2 - 12)^2 + x_3^4 + 3(x_4 - 11)^2 + 10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7 \quad (5.14)$$

sujeto a:

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

donde $-10 \leq x_i \leq 10$ ($i = 1, \dots, 7$). La solución óptima está representada por el punto $\mathbf{x} = (2,330499, 1,951372, -0,47775414, 4,365726, -0,6244870, 1,038131, 1,594227)$ donde $f(\mathbf{x}) = 680,6300573$. Las restricciones activas en este punto son: g_1 y g_4 .

■ G10

Problema propuesto en [169] y consiste en la minimización de la siguiente función:

$$f(\mathbf{x}) = x_1 + x_2 + x_3 \quad (5.15)$$

sujeto a:

$$\begin{aligned} g_1(\mathbf{x}) &= -1 + 0,0025(x_4 + x_6) \leq 0 \\ g_2(\mathbf{x}) &= -1 + 0,0025(x_5 + x_7 - x_4) \leq 0 \\ g_3(\mathbf{x}) &= -1 + 0,01(x_8 - x_5) \leq 0 \\ g_4(\mathbf{x}) &= -x_1x_6 + 833,33252x_4 + 100x_1 - 83333,333 \leq 0 \\ g_5(\mathbf{x}) &= -x_2x_7 + 1250x_5 + x_3x_4 - 1250x_4 \\ &\leq 0 \\ g_6(\mathbf{x}) &= -x_3x_8 + 1250000 + x_2x_5 - 2500x_5 \leq 0 \end{aligned}$$

donde $100 \leq x_1 \leq 10000$, $1000 \leq x_i \leq 10000(i + 2, 3)$, y $10 \leq x_i \leq 1000(i = 4, \dots, 8)$. La solución óptima en este punto es $\mathbf{x} = (579,3167, 1359,943, 5110,071, 182,0174, 295,5985, 217,9799, 236,4162, 395,5979)$ para el cual $f(\mathbf{x}) = 7049,3307$. Las restricciones g_1, g_2 , y g_3 están activas en este punto.

■ G11

Problema propuesto en [104] y consiste en la minimización de la función:

$$f(\mathbf{x}) = x_1^2 + (x_2 - 1)^2 \quad (5.16)$$

sujeto a:

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

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

■ G13

Problema propuesto en [169] y consisten en la minimización de la función:

$$f(\mathbf{x}) = e^{x_1 x_2 x_3 x_4 x_5} \quad (5.17)$$

sujeto a:

$$h_1(\mathbf{x}) = x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 - 10 = 0$$

$$h_2(\mathbf{x}) = x_2 x_3 - 5 x_4 x_5 = 0$$

$$h_3(\mathbf{x}) = x_1^2 + x_2^3 + 1 = 0$$

donde $-2,3 \leq x_i \leq 2,3(i = 1, 2)$ y $3,2 \leq x_i \leq 3,2(i = 3, 4, 5)$. La solución óptima en este punto es $\mathbf{x} = (-1,777143, 1,595709, 1,827247, 0,7636413, -0,763645)$ y $f(\mathbf{x}) = 0,0539498$.

■ G14: Función de Floudas-Pardalos (#2)

Este problema fue obtenido de Floudas et al. [58] y consiste en maximizar:

$$F_{FP_2}(\mathbf{x}) = -x_1 - x_2,$$

sujeto a:

$$\begin{aligned} x_2 &\leq 2x_1^4 - 8x_1^3 + 8x_1^2 + 2 \\ x_2 &\leq 4x_1^4 - 32x_1^3 + 88x_1^2 - 96x_1 + 36, \end{aligned}$$

y límites, $0 \leq x_1 \leq 3$ y $0 \leq x_2 \leq 4$. La mejor solución conocida es $(x_1, x_2)^* = (2,3295, 3,1783)$ y el valor de la función en aquel punto es $-5,5079$. La Figura 5.20 muestra el espacio de búsqueda factible determinado por las restricciones de desigualdad y la posición aproximada del mejor valor conocido para esta función.

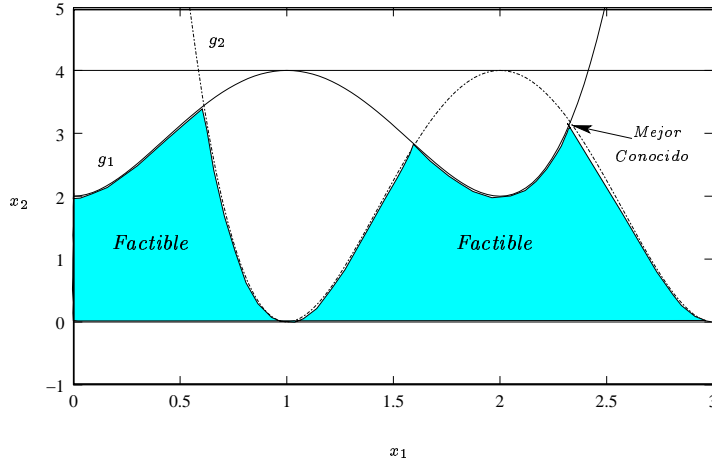


Figura 5.20: Posición aproximada del mejor valor conocido en el espacio de búsqueda factible determinado por las restricciones g_1 y g_2 .

■ G15: Función de Soland

Este problema fue obtenido de Floudas et al. [58]. Este caracteriza un función convexa sujeta a restricciones de igualdad no lineales. El problema consiste en minimizar:

$$F_S(\mathbf{x}) = -12x_1 - 7x_2 + x_2^2,$$

sujeto a:

$$-2x_1^4 + 2 - x_2 = 0$$

con rangos, $0 \leq x_1 \leq 2$ y $0 \leq x_2 \leq 3$. La mejor solución conocida es $(x_1, x_2)^* = (0,71751, 1,470)$ y el valor de la función en este punto es $-16,73889$. La figura 5.21 muestra el mejor punto conocido el cual está sobre la frontera del espacio de búsqueda factible.

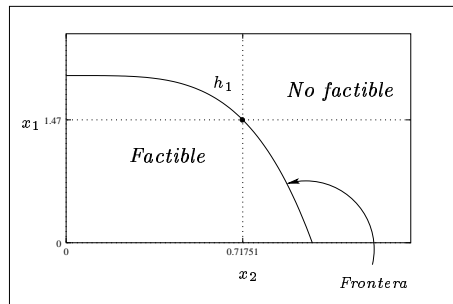


Figura 5.21: El mejor punto conocido está sobre el espacio de búsqueda determinado por la restricción de igualdad h_1

■ G16: Función de Baluja's (#1)

Este problema fue propuesto por Baluja en [14] y consiste en maximizar:

$$F_{B_1}(\mathbf{x}) = \frac{100}{0,00001 + \sum_{i=1}^n |y_i|}$$

donde $y_i = x_i + y_{i-1}$ para $i = 1, \dots, n$ con $y_0 = 0$,

sujeto a

$$\sum_{i=1}^n x_i^2 = 1$$

Sobre la esfera con coordenadas positivas, la función tiene una solución global $(x_1, \dots, x_n) = (0, \dots, 1)$ y el valor de la función en dicho punto es 99,999.

■ G17: Función de Baluja (#2)

Este problema involucra la misma función objetivo que F_{B_1} :

$$F_{B_2}(\mathbf{x}) = \frac{100}{0,00001 + \sum_{i=1}^n |y_i|}$$

sujeto a

$$\sum_{i=1}^n x_i^2 = 1$$

excepto que $y_i = x_i + \sin(y_{i-1})$ para $i = 1, \dots, n$ con $y_0 = 0$.

De la misma manera que la función F_{B_1} , esta función tiene una solución global en el punto $(x_1, \dots, x_n) = (0, \dots, 1)$ y el valor de la función en ese punto es 99,999.

5.7.2. Variables Discretas

■ The MKP

El Multiple Knapsack Problem (MKP) puede ser formulado como sigue [67]:

$$\begin{aligned} &\text{maximizar } \sum_{j=1}^n p_j x_j \\ &\text{sujeto a } \sum_{j=1}^n r_{ij} x_j \leq c_i \quad i = 1, \dots, m, \\ &x_j \in \{0, 1\} \quad j = 1, \dots, n. \end{aligned} \tag{5.18}$$

Existen m restricciones en este problema, de manera tal que MKP es también denominado el *m-dimensional Knapsack Problem*. Sea $I = \{1, \dots, m\}$ y $J = \{1, \dots, n\}$, con $c_i \geq 0$ para todo $i \in I$. Un MKP *bien-formulado* asume que $p_j > 0$ y $r_{ij} \leq c_i \leq \sum_{j=1}^n r_{ij}$ para todo $i \in I, j \in J$, dado que la violación de cualquiera de estas condiciones

dará como resultado que algún x_j pueda ser cero o que algunas restricciones sean eliminadas. Hay que notar que la matriz $[r_{ij}]_{m \times n}$ y el vector $[c_i]_m$ son no negativos, lo cual distingue a este problema del problema general 0-1 de programación lineal entera. Muchos problemas de importancia práctica pueden ser formulados a través de MKP. Por ejemplo, el problema de distribución de presupuesto donde un proyecto j tiene un beneficio p_j y consume r_{ij} unidades del recurso i . El objetivo es encontrar un subconjunto de proyectos $S^* \subseteq J$ tales que el beneficio total es maximizado y todas las restricciones sobre los recursos sean satisfechas.

Ejemplo de una instancia de MKP Sea $n = 4$ y $m = 3$ el número de componentes y restricciones respectivamente. Los siguientes vectores y la matriz de la figura 5.22 representan los valores para p_j , c_i , y r_{ij} respectivamente: $(p_1, p_2, p_3, p_4) = (4, 10, 2, 6)$, $(c_1, c_2, c_3) = (8, 12, 10)$.

Los siguientes conjuntos representan soluciones factibles para esta instancia. Para cada solución se muestra el beneficio respectivo:

$$\begin{aligned} S_1 &= \{1, 3\}, & profit(S_1) &= 6 \\ S_2 &= \{2, 4\}, & profit(S_2) &= 16 \\ S_3 &= \{2, 3, 4\}, & profit(S_3) &= 18 \end{aligned}$$

$$\begin{pmatrix} r_{11} & r_{12} & r_{13} & r_{14} \\ r_{21} & r_{22} & r_{23} & r_{24} \\ r_{31} & r_{32} & r_{33} & r_{34} \end{pmatrix} = \begin{pmatrix} 4 & 4 & 2 & 1 \\ 6 & 6 & 3 & 3 \\ 4 & 4 & 2 & 2 \end{pmatrix}$$

Figura 5.22: Matriz que representa los recursos.

■ SCP

El problema de Cobertura de Conjunto o “Set Covering Problem”

(SCP) es definido de la siguiente manera [67]:

$$\begin{aligned}
 & \text{minimize ar } \sum_{j=1}^n c_j x_j \\
 & \text{sujeto a} \\
 & \sum_{j=1}^n a_{ij} x_j \geq 1 \quad i = 1, \dots, m, \\
 & x_j \in \{0, 1\} \quad j = 1, \dots, n.
 \end{aligned} \tag{5.19}$$

En otras palabras, es necesario seleccionar un subconjunto de columnas de una matriz binaria $A = (a_{ij})$ para minimizar el costo total y que al mismo tiempo satisfagan las restricciones de la cobertura de las filas, es decir que todas las filas deben estar cubiertas por al menos una columna incluida en la solución.

Formalmente, sea $J = \{1, \dots, n\}$ el conjunto de índices de las columnas de la matriz A ; el objetivo es encontrar un conjunto $S^* \subseteq J$ tal que $\forall S' \subseteq J$, $\sum_{j \in S^*} c_j \leq \sum_{j \in S'} c_j$ y que al mismo tiempo todas las restricciones sean satisfechas.

Ejemplo de una instancia de SCP

Sea la matriz de la Figura 5.23 y el vector de costos $c = (5, 3, 7, 2, 8, 4, 4, 3, 5, 2)$ una instancia de SCP.

$$\begin{array}{cccccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
 A = & \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} & \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{matrix}
 \end{array}$$

Figura 5.23: Ejemplo de una instancia de SCP (la matriz binaria).

Los siguientes conjuntos representan soluciones factibles para esta instancia. Para cada solución calculamos su respectivo costo:

$$S_1 = \{4, 5, 9, 10\}, \quad \text{cost}(S_1) = 17$$

$$S_2 = \{1, 3, 5, 10\}, \quad \text{cost}(S_2) = 22$$

$$S_3 = \{3, 4, 5, 10\}, \quad \text{cost}(S_3) = 14$$

- **MISP** El Problema del Máximo Conjunto Independiente o “Maximum Independent Set Problem” (MISP) se define de la siguiente manera:

Dado un grafo $G = \langle V, E \rangle$, el problema consiste en determinar el conjunto de vértices $S^* \subseteq V$ de mayor cardinalidad con la restricción que ninguno de los vértices en S^* sean adyacentes [67].

Ejemplo de una instancia de MISP

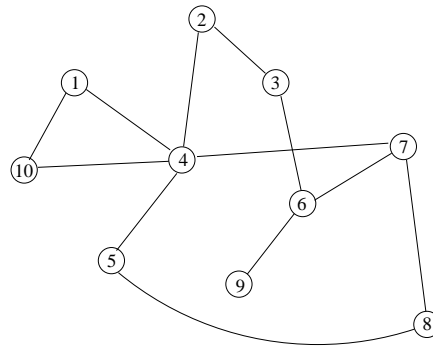


Figura 5.24: Instancia de MISP de tamaño 10.

El grafo en la Figura 5.24 representa una instancia pequeña de MISP donde $|V| = 10$. Algunas soluciones factibles para esta instancia y su respectivo valor de la función objetivo son:

$$S_1 = \{2, 5, 7, 9, 10\} \text{ con } |S_1| = 5$$

$$S_2 = \{3, 4, 8, 9\} \text{ con } |S_2| = 4$$

$$S_3 = \{1, 2, 6, 8\} \text{ con } |S_3| = 4$$

Capítulo 6

Ant Colony Optimization y operadores de frontera

6.1. Introducción

Es este capítulo se presenta el estudio experimental de la aplicación de algoritmos ACO para el conjunto completo de problemas de optimización presentados en el capítulo anterior (ver sección 5.7). De aquí en adelante, dichos algoritmos serán referidos como SH_C y SH_D para problemas continuos y discretos, respectivamente.

Es importante destacar que para los problemas continuos, existe una representación concreta de las soluciones y ésta está dada por un vector de punto flotante, mientras que para los problemas discretos no existe una representación explícita de las soluciones, dado que las características del método para este tipo de problemas propone la construcción de las soluciones de una manera paso-a-paso. En síntesis, para los problemas continuos, el proceso de aprendizaje del algoritmo está orientado a determinar en forma directa las zonas o regiones del espacio de búsqueda más prometedoras. En el caso de los problemas discretos, el aprendizaje está centrado en las componentes del problema a escoger por un agente durante el proceso de construcción. De esta manera también se pretende alcanzar las regiones más prometedoras, pero en forma indirecta.

La aplicación de SH_C y SH_D a cada uno de los problemas estudiados,

puede requerir un mayor o menor número de ajustes. En general, los ajustes necesarios son mínimos dada la robustez el método para manejar, sin mayores cambios, una gran diversidad de problemas. Por ejemplo, para los problemas continuos sólo se modifican las estructuras y valores que tienen que ver específicamente con el problema, esto es, función objetivo, número de variables y las restricciones del mismo incluyendo el rango de cada una de las variables. En el caso de los problemas discretos, sólo es necesario modificar la heurística asociada al problema y las condiciones que controlan la factibilidad de la solución.

Dado que SH_C puede ser aplicado a problemas con más de una restricción, es claro que es necesario usar algún esquema complementario para el manejo de restricciones, aunque esto no es necesario cuando el problema tiene sólo una restricción del tipo $h(x)$ o bien una única del tipo $g(x)$ la cual está activa en el óptimo. La técnica complementaria usada para el manejo de restricciones es la técnica de penalización¹ incluida en la siguiente función de mérito para las soluciones:

$$\phi(x, \mu) = f(x) + \mu(t) \left(\sum_{j=1}^q \max\{0, g_j(x)\} + \sum_{j=q+1}^m |h_j(x)| \right) \quad (6.1)$$

donde $\mu(t)$ es un factor de penalización que puede variar o no acorde al avance de las iteraciones del algoritmo. Es decir, $\mu(t) = \mu_0$ para todo t , o bien puede ir cambiando a medida que las iteraciones avanzan. En general el cambio es tal que $\mu(0) \leq \mu(1) \leq \mu(2) \dots \leq \mu(t_{max})$; esto implica la intensificación de la búsqueda acotada a zonas factibles del espacio de búsqueda.

Es importante destacar que la técnica elegida para el tratamiento de soluciones no factibles es muy simple. La idea aquí es evitar incorporar técnicas de manejo de restricciones muy avanzadas que puedan influir de alguna manera en el enfoque de frontera propuesto. Esto podría llevar a no saber claramente a quién atribuir eventuales beneficios de la

¹Todos los problemas fueron tratados como de maximización

aplicación del método. Cabe recordar que el enfoque de frontera en este contexto implica la generación de una solución sobre la frontera de una única restricción a la vez, lo que hace necesario tener en cuenta a) cuál restricción elegir y 2) el resto de las restricciones (si existen) para determinar si son violadas o no.

Por otro lado, el algoritmo SH_D para problemas discretos del tipo estudiado en esta tesis, merece una consideración especial. Tal como fuera explicado en la sección 5.6, un algoritmo ACO para problemas de subconjunto involucra una población de agentes que construyen en forma independiente una solución paso-a-paso. Cada solución, al final de proceso de construcción, es una solución factible que está sobre la frontera entre el espacio de búsqueda factible y no factible. Por lo tanto el método *per se*, tal como fuera planteado, nunca genera soluciones no factibles que eventualmente deberían ser penalizadas (recordar la visualización del espacio de búsqueda y las zonas factibles según la figura 5.1). A diferencia de los problemas continuos, para los problemas discretos es necesario disponer de un conjunto de instancias de los mismos para la aplicación de un determinado algoritmo. En este sentido, las instancias consideradas proceden desde repositorios específicamente creados para ser usados como *benchmarks* para la evaluación de los algoritmos propuestos en la comunidad de investigadores (por ejemplo, el repositorio de la OR Library mantenido por J. Beasley [20] y el repositorio de DIMACS [43]); otro tipo de instancias, son el producto de la aplicación de procedimientos específicamente diseñados que incluyen forma sistemáticas y/o aleatorias de generación. Para cada problema en particular se mencionará la procedencia de las respectivas instancias cuando corresponda.

A fin de obtener estimadores más confiables de la media y desviación estándar, los algoritmos se ejecutaron 30 veces por cada problema (instancia) considerada. Esta observación se mantiene para los experimentos realizados con el enfoque evolutivo de frontera (próximo capítulo).

En las siguientes secciones se presentan los resultados para los problemas continuos y luego los discretos. Para cada uno de los problemas se realiza un análisis de los resultados y comparaciones indirectas con resultados de algoritmos representantes del estado del arte. En el caso de los problemas discretos, se muestra adicionalmente para cada uno de ellos la heurística (η) aplicada a cada uno de los problemas estudiados (MKP, SCP y MISP).

El principal objetivo del estudio experimental es mostrar la factibilidad del enfoque de frontera en el contexto de los algoritmos ACO para clases de problemas y poder establecer la calidad de los resultados alcanzados en comparación con los resultados reportados usando otros enfoques. Otro punto a destacar en el contexto del estudio experimental, es que muchos de los valores de los parámetros usados corresponden a valores sugeridos o bien que han mostrado ser efectivos en aplicaciones anteriores. Por ejemplo, a partir de los primeros experimentos de Dorigo et al. [39] para el Problema de Viajante de Comercio y otros problemas de optimización combinatoria, surgen valores estándar para la importancia del rastro ($\alpha = 1$), para la importancia de la heurística ($\beta = 5$) y el coeficiente de persistencia (ρ alrededor de 0,5), etc. Sin embargo, esto no son reglas a seguir y dichos valores pueden variar según el tipo de problema y de instancia del problema.

6.2. Problemas continuos (aplicación de SH_C)

Aquí se muestran algunos resultados obtenidos para los casos de prueba $G01$, $G02$, $G03$, $G04$, $G05$, $G06$, $G07$, $G09$, $G10$, $G11$, $G13$, $G14$, $G15$, $G16$, y $G17$ según se describieron en la sección 5.7.1. Los problemas $G08$ y $G12$, ampliamente conocidos y también extensivamente estudiados, no fueron estudiados aquí dado que ninguna de sus restricciones están activas en los puntos óptimos o mejores conocidos.

Para todos los problemas, se muestran los resultados de la aplicación

de SH_C con los siguientes parámetros: 50 hormigas, 20 direcciones de búsqueda (ver sección 5.6) y 30000 ciclos. Dado que aquí no se usa información local (heurística η en la ecuación 5.3) no es necesario usar los parámetros α y β , y por ende darle algún valor. En este sentido, SH_D sólo usa valores globales dados por el rastro depositado en cada una de las direcciones de búsqueda.

En cada una de las siguientes tablas se muestra el nombre del problema y su valor óptimo (o mejor conocido) con excepción del problema G2, para el cual se muestran los resultados según variaciones del número de variables (problema escalable) y los respectivos óptimos conocidos son descritos en cada una de las tablas. La columna “Restricción” indica el número de la restricción sobre la cual el algoritmo realizó la búsqueda, independientemente de saber o no que la misma está activa en el óptimo. Existen además dos rótulos adicionales sobre esta columna indicados por “Activas” y “Todas”. Los resultados correspondientes a cada una de esas filas está vinculado con alguna de las posibilidades de aplicación del algoritmo que se mencionaron en la sección 4.3.1. Más precisamente, el rótulo “Activas” indica que el algoritmo realizó la búsqueda sobre las restricciones activas de a una por vez y cambiando cada T número de ciclos. Claramente, esto se realiza considerando resultados previos de otros métodos que indican que ciertas restricciones están activas en el óptimo y esto nos permitiría explotar dicho conocimiento. Otra situación que no necesariamente requiere de alguna experiencia previa es cuando el problema consta de sólo restricciones de igualdad ($h_j(x)$). En este caso todas las restricciones estarán activas en el punto óptimo, sea éste conocido o no. Así, el algoritmo comienza a realizar la búsqueda sobre una restricción activa en particular y luego de T iteraciones y cíclicamente considera la próxima restricción activa. Es decir, para un problema que tiene activas las restricciones 1, 5, y 6, el algoritmo podría comenzar por la 1, luego la 5, y 6, y vuelve nuevamente a la 1 y así siguiendo hasta completar un número específico de iteraciones. Sin embargo, cuando

no se tiene información del problema acerca de las posibles restricciones activas y no todas son restricciones por igualdad, es posible también, aplicar el método de manera tal que considere cíclicamente cada una de las restricciones del problema para realizar la búsqueda. En este caso, los resultados correspondientes están rotulados con “Todas” para indicar dicha forma de realizar la búsqueda. Esto último tiene por objetivo mostrar que el método aún puede ser eficaz sin tener un conocimiento previo de las posibles restricciones activas en el punto óptimo.

Con respecto a las diferentes maneras de realizar la búsqueda, se podrían eliminar del problema aquellas restricciones que no están activas en el mejor punto conocido, asumiendo que se ha detectado esta situación a través de estudios o análisis previos. Se dice que estas restricciones (Nocedal et al.[91]) no son significativas o bien, están inactivas el punto óptimo (figura 6.1). Por ello, éstas podrían ser dejadas fuera de consideración en el diseño de algún algoritmo para resolver el problema. Sin embargo, no hemos considerado esta situación en nuestro estudio experimental a los efectos de evaluar el desempeño de los algoritmos, usando todas las opciones de búsqueda, independientemente del conocimiento previo acerca de las restricciones activas.

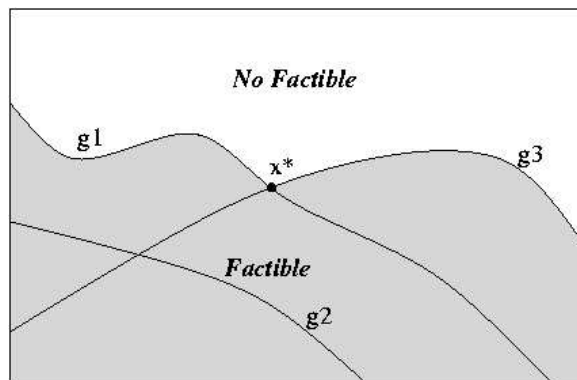


Figura 6.1: Un problema hipotético con tres restricciones ($g_i(x)$ con $i = 1, 2, 3$). La restricción g_2 no está activa en el óptimo (x^*), por lo tanto no es relevante para ser considerada en el proceso de búsqueda.

Siguiendo con la descripción de la tabla 6.1, la columna #Factibles

muestra el número de soluciones factibles encontradas (conteo de la mejor solución en cada corrida). La columna Prom($\#E$) muestra el número promedio de evaluaciones necesarias para encontrar el mejor valor en cada corrida. El número T a partir del cual el algoritmo cambia de restricción (esto es cuando son elegidas las opciones de búsqueda “Activas” y “Todas”) fue establecido en 200 para estos experimentos. Por otro lado, los valores de $\mu(t)$, para $t = 0, 1, \dots, t_{max}$, fueron determinados empíricamente para cada uno de los problemas considerados. En la mayoría de los casos se usó un valor fijo para el factor de penalización, es decir, $\mu(t) = \mu_0$, para $t = 0, 1, \dots, t_{max}$. Los valores y modalidad de la variación de este factor serán mostrados por cada problema (en el encabezado de las tablas de resultados respectivas). Es importante destacar que para los problemas con una sólo restricción, no es necesario incorporar un factor de penalización. Esta observaciones, para el factor de penalización, se mantienen para los experimentos realizados con EAs cuyos resultados son mostrados en el próximo capítulo.

Tabla 6.1: Valor de penalización usado para G01 es $\mu(t) = 1000$ para $t = 0, 1, \dots, t_{max}$.

Restricción	Función G01 (-15.00)					
	Mejor	Media	DesvEstand	Peor	# Factibles	Prom($\#E$)
1	-15.00	-14.99	0.001	-14.996	30	274800
2	-15.00	-14.96	0.012	-14.995	30	159720
3	-15.00	-14.99	0.001	-14.965	30	381800
4	-14.27	-13.54	.38	-13.18	29	544400
5	-13.84	-13.48	0.32	-13.04	25	433800
6	-14.22	-13.39	0.47	-13.00	26	407200
7	-15.00	-14.78	0.2	-14.65	26	213400
8	-15.00	-14.74	0.49	-14.46	27	723400
9	-15.00	-14.67	0.76	-13.08	30	454800
Activas	-15.00	-15.00	0	-15.00	30	81400
Todas	-15.00	-15.00	0	-15.00	30	104000

Problema G01

Las filas 1 a 9, de la tabla 6.1 indican la restricción elegida para realizar la búsqueda. Vemos que los mejores resultados son obtenidos cuando dicha búsqueda se realiza sobre alguna de las restricciones activas en el

óptimo. Sin embargo, su comportamiento se puede considerar satisfactorio cuando se consideran las restricciones 4,5 y 6. En el caso de realizar la búsqueda por la frontera de las restricciones activas únicamente (fila “Activas”), vemos que en todos los casos se alcanza el óptimo y que todas las soluciones encontradas son factibles. Una situación similar ocurre cuando todas las restricciones son consideradas (fila ‘Todas’). Como ya se mencionó, esta opción se puede realizar cuando, suponiendo que al menos una de las restricciones esté activa en el óptimo, no se sabe exactamente cuáles son las activas. Así, el algoritmo va cíclicamente saltando de una restricción a otra, a medida que la búsqueda avanza. El número de evaluaciones requeridas para encontrar el mejor valor disminuye notablemente cuando la búsqueda se restringe a las restricciones activas únicamente. Sin embargo, dicha disminución también se aprecia realizando la búsqueda sobre todas las restricciones, lo que en general es esperable si se tiene en cuenta que las activas están incluidas en la búsqueda de las soluciones.

Tabla 6.2: Aquí se muestran los valores encontrados para la función de Keane con $n = 20, 50$ y 100 variables. Los mejores valores conocidos para cada uno de ellos son $0,803619$ [83, 107], $0,831937$ [157] y ND respectivamente, donde ND indica resultado no disponible.

Nro. de Variables	Función G02 (*)					
	Mejor	Media	DesvEstand	Peor	# Factibles	Prom(#E)
20	0.8036190867	0.8025656939	0.0032	0.7930839658	30	29500
50	0.8352618814	0.8339309692	0.0021	0.8259508014	30	35900
100	0.8456841707	0.8446936011	0.0007	0.8423509002	30	46700

Problema G02

Cabe aclarar que para este problema sólo se muestra la opción “Activas”, ya que se agruparon los resultados para 3 valores de n . Si bien se realizaron experimentos para realizar búsqueda por la restricción 2 y “Todas” (es decir 1 y 2), los resultados fueron desestimados. La razón es que la frontera de la segunda restricción está muy lejos del mejor conocido. Además tampoco fue necesario preocuparse por los valores de $\mu(t)$

cuando se realiza la búsqueda por la restricción 1. La razón aquí es que cualquier solución sobre la frontera de la restricción 1 satisface la 2, por lo tanto no tiene sentido considerar valores para penalizar.

El problema $G02$ es interesante de destacar dado que para $n = 20$ la mayoría de los métodos para manejo de restricciones no han podido resolver este problema, al menos hasta el mejor conocido [151]. Vemos en la tabla 6.2 los resultados encontrados para distintos valores de n , siendo $n = 20$ el más ampliamente usado como *benchmark* en los estudios de técnicas para manejo de restricciones. Los resultados muestran la efectividad de SH_C para resolver este problema. Aunque no mostrados en la tabla, 28 de las 30 soluciones encontradas corresponden al valor objetivo 0,8036190867, y 2 al valor 0,7930839658, de ahí los respectivos valores del promedio y desviación. Asimismo se agregan dos filas más para los valores de $n = 50, 100$. Para $n = 50$, el mejor valor reportado es 0,8331937 [157], mientras que SH_C obtuvo 0,8352618814. A modo de ejemplo, se consideró el valor de $n = 100$ para el cual el algoritmo arrojó 0,8456841707 como mejor valor encontrado. También es importante resaltar que la totalidad de las soluciones encontradas fueron factibles. El número promedio de evaluaciones para resolver $G02$ es elevado, pero hay que considerar algunas particularidades de las soluciones encontradas. Por ejemplo, para $n = 20$, el mejor valor reportado es 0,803619, dando una precisión de 6 dígitos decimales. Si el número de evaluaciones sólo es considerado hasta alcanzar el mejor valor dentro de dicho número de dígitos, este se reducirá en forma considerable como lo demuestran algunos experimentos adicionales, principalmente para $n = 20$.

Tabla 6.3: $G03$ tiene una restricción $h(x)$ y no es necesario asignar valores a μ . Se muestran los resultados para $n = 20, 50$ variables

Nro. de Variables	Función $G03$ (1.0)					
	Mejor	Media	DesvEstand	Peor	# Factibles	Prom(#E)
20	1.0	1.0	0.0	1.0	30	140000
50	1.0	1.0	0.0	1.0	30	389500

Problema G03

Similarmente al caso G02, en la tabla 6.3 se muestran los resultados para $n = 20, 50$ por ser un problema escalable. El óptimo en cualquiera de los dos casos es 1. Claramente se puede observar el buen desempeño del enfoque para este problema con sólo una restricción de igualdad. El número de evaluaciones crece a medida que crece el número de variables de la solución. De cualquier manera, es importante destacar que el algoritmo converge rápidamente a un valor muy cercano al óptimo y la mayor parte del tiempo restante, es ocupada en realizar ajustes muy pequeños para alcanzar el óptimo conocido con un mayor grado de precisión. La figura 6.2 muestra el comportamiento del algoritmo para $n = 20$ en donde se puede observar que antes de la iteración número 100 el mejor valor encontrado está muy próximo al punto óptimo, con una diferencia menor a 0,00001. Dicha figura muestra el promedio de los valores objetivo de 30 corridas del algoritmo y para $n = 20$. Un comportamiento similar es observado para los restantes valores de n considerados.

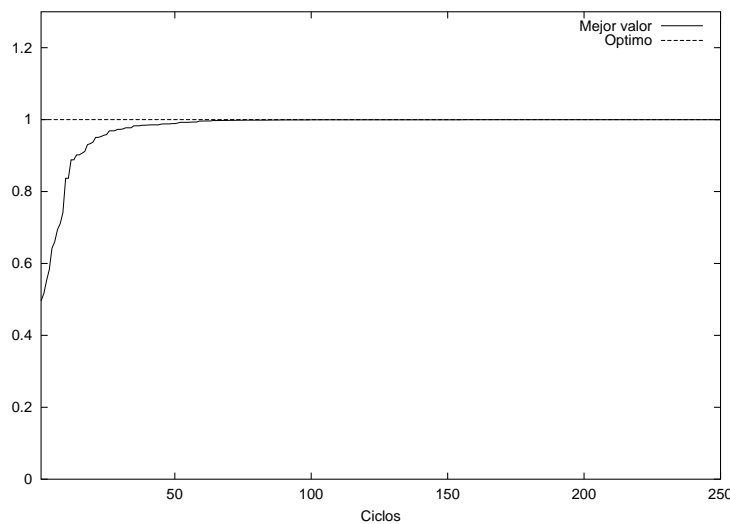


Figura 6.2: Convergencia de los mejores valores en cada generación. Se observa una rápida convergencia del algoritmo a valores muy cercanos al óptimo

Problema G04

Este caso presentó algunos inconvenientes al ser aplicado sobre las

Tabla 6.4: Valor de penalización usado para G04 es $\mu(t) = 800000$ para $t = 0, 1, \dots, t_{max}$.

Restricción	Función G04 (-30655.539)					
	Mejor	Media	DesvEstand	Peor	# Factibles	Prom(#E)
1	-30665.539	-30665.357	0.04	-30665.157	30	512200
2	-	-	-	-	-	-
3	-	-	-	-	-	-
4	-	-	-	-	-	-
5	-	-	-	-	-	-
6	-30655.539	-30665.302	0.01	-30665.290	30	326400
Activas	-30655.539	-30655.539	0.001	-30655.539	30	19800
Todas	-	-	-	-	-	-

restricciones 2, 3, 4 y 5, que son justamente aquellas restricciones que no están activas en el óptimo. El principal inconveniente está vinculado con la imposibilidad del algoritmo de encontrar soluciones sobre la frontera de cada una de dichas restricciones al realizar la búsqueda. Dada esta situación, no fue posible completar la fila “Todas” por las razones previamente expuestas (ver tabla 6.4). De cualquier manera, el algoritmo fue capaz de encontrar la mejor solución cuando realizó la búsqueda ya sea por alguna de las dos activas (restricciones 1 y 6) o bien alternando entre las dos (fila “Activas”). Es interesante resaltar que en este último caso, no sólo se redujo la desviación estándar, sino también el número promedio de evaluaciones, indicando una aceleración en la convergencia para este problema en particular, cuando el algoritmo solamente realiza la búsqueda sobre las activas.

Tabla 6.5: Valor de penalización usado para G05 es $\mu(t) = 10$ para $t = 0, 1, \dots, t_{max}$.

Restricción	Función G05 (5126.49)					
	Mejor	Media	DesvEstand	Peor	# Factibles	Prom(#E)
1	-	-	-	-	-	-
2	-	-	-	-	-	-
3	5126.50	5133.29	9.284	5147.81	6	100800
4	5126.51	5134.70	11.219	5164.91	11	340000
5	5126.68	5130.55	3.656	5136.08	11	180000
Activas	5126.50	5138.37	8.20	5132.14	6	94000
Todas	5126.50	5143.77	10.60	5163.56	5	135800

Problema G05

Las dos primeras filas de la tabla 6.5 no contienen valores dado que

el algoritmo no fue capaz de encontrar ninguna solución factible en las 30 corridas. Dichas filas corresponden a las restricciones de desigualdad (g_i). Sin embargo, siguiendo cualquiera de las restricciones de igualdad (h_i), el algoritmo es capaz de alcanzar varias soluciones factibles de muy buena calidad. La aplicación del algoritmo sobre las restricciones activas y el conjunto total arrojó muy buenos valores que podrían ser considerados como óptimos. Sin embargo, este caso presenta cierta dificultad para el algoritmo evidenciado por el escaso número de soluciones factibles encontradas en cada caso. En relación al número de evaluaciones, también se observa aquí una disminución de las mismas cuando se realiza la búsqueda sobre la restricciones activas, aunque la disminución no es muy importante si se considera la ejecución del algoritmo sobre la restricción 3.

Tabla 6.6: Valor de penalización usado para G06 es $\mu(t) = 10000$ para $t = 0, 1, \dots, t_{max}$.

Restricción	Función G06 (6961.81)					
	Mejor	Media	DesvEstand	Peor	# Factibles	Prom(#E)
1	-6961.79	-6961.71	0.075	-6169.54	11	122600
2	-6961.81	-6961.72	0.097	-6961.34	25	103000
Activas	-6961.81	-6961.74	0.070	-6961.71	25	80000

Problema G06

Este es un caso en donde todas las restricciones son igualdades (h_i) y en consecuencia la fila “Todas” es equivalente a la fila “Activas” (ver tabla 6.6). El mejor comportamiento del algoritmo se refleja en los resultados de la restricción 2 y las activas. No sólo por la calidad de los valores alcanzados, sino que también por el número de soluciones factibles encontradas. Adicionalmente, se produce como en la mayoría de los casos, una disminución en el número de soluciones evaluadas al usar la búsqueda combinada de todas las restricciones activas.

Problema G07

Nuevamente aquí (tabla 6.7) se evidencia la disminución de la calidad de los resultados obtenidos cuando el algoritmo sólo busca por las res-

Tabla 6.7: Valor de penalización usado para G07 es $\mu(t) = 20000$ para $t = 0, 1, \dots, t_{max}$.

Restricción	Función G07 (24.306)					
	Mejor	Media	DesvEstand	Peor	# Factibles	Prom(#E)
1	24.37	29.59	4.83	42.97	30	70000
2	24.51	35.10	23.02	121.56	30	133600
3	24.56	28.31	5.54	50.83	30	83600
4	24.79	54.17	70.46	380.03	30	83600
5	24.52	34.52	16.39	77.19	30	85000
6	24.79	31.12	6.46	48.40	30	720800
7	33.08	38.86	4.01	46.53	30	71200
8	41.03	46.86	20.92	127.06	30	260200
Activas	24.37	24.64	0.15	24.92	30	35600
Todas	24.38	24.76	0.16	25.22	30	56000

tricciones que no están activas en el óptimo (restricciones 7 y 8). En el resto de los experimentos al algoritmo encontró muy buenas soluciones y factibles en cada una de las corridas. Es de destacar la restricción 1 como la más ventajosa cuando se aplica en forma individual. De cualquier manera, el enfoque de búsqueda sobre las activas únicamente es a partir del cual se obtienen los mejores resultados, pero no distan demasiado de la calidad de los mismos en su aplicación sobre todas las fronteras (ver Mejor, Peor y DesvEstand). La ejecución del algoritmo con la opción sobre todas las activas permite una ganancia en la velocidad de convergencia y por tanto en el número promedio de evaluaciones para alcanzar el mejor valor.

Tabla 6.8: Valor de penalización usado para G09 es $\mu(t) = 2000$ para $t = 0, 1, \dots, t_{max}$.

Restricción	Función G09 (680.63)					
	Mejor	Media	DesvEstand	Peor	# Factibles	Prom(#E)
1	680.63	680.66	0.10	681.29	30	80400
2	1664.00	1890.01	119.92	1982.72	5	108000
3	840.00	880.82	15.06	890.56	29	22200
4	680.64	680.96	0.96	681.95	29	43000
Activas	680.63	680.67	0.026	680.72	30	7400
Todas	680.65	680.75	0.056	680.89	30	19400

Problema G09

El problema G09 tiene dos restricciones activas en el óptimo, éstas son la 1 y 4. Los resultados obtenidos para las mismas (tabla 6.8) dista mucho en calidad cuando es comparado con aquellos obtenidos a partir

de la búsqueda sobre las restricciones 2 y 3 como puede observarse en la tabla respectiva. Sin embargo, las dos últimas filas ponen en evidencia la utilidad de la búsqueda alternando entre distintas restricciones en una misma ejecución (sólo activas o todas). También es de destacar el número de soluciones factibles encontradas y el número reducido de evaluaciones realizadas cuando se consideran sólo las restricciones activas.

Tabla 6.9: Valor de penalización usado para G10 es $\mu(t) = 1,05 * \mu(t - 1)$ para $t = 0, 1, \dots, t_{max}$, con $\mu(0) = 200000$

Restricción	Función G10 (7049.331)					
	Mejor	Media	DesvEstand	Peor	# Factibles	Prom(#E)
1	7101.50	7346.61	202.15	7682.20	9	147700
2	7063.02	8169.68	1866.32	10325.00	3	131600
3	7057.27	7406.51	148.60	7518.91	9	148600
4	7095.27	7349.83	360.00	7604.39	2	128200
5	-	-	-	-	-	-
6	-	-	-	-	-	-
Activas	7052.30	7199.01	175.01	7943.15	30	42800
Todas	7068.04	7141.87	52.27	7239.54	30	9800

Problema G10

El problema G10 es uno de los más difíciles, evidenciado principalmente por los resultados obtenidos según estudios previos. Esta situación no es distinta para nuestro algoritmo. Un análisis de la tabla 6.9 muestra que para las restricciones 5 y 6 no fue posible encontrar ninguna solución factible. Para el resto de las opciones el algoritmo encontró buenas soluciones muy cercanas al óptimo. Sin embargo, la búsqueda sobre una única restricción (casos 1,2,3 y 4) no arroja un gran porcentaje de soluciones factibles si dicho número es comparado con el número de soluciones factibles encontradas siguiendo el esquema “Activas” y “Todas”. Dichas opciones de búsqueda muestran además, una importante aceleración en la convergencia. De estas dos últimas, la búsqueda sobre todas las restricciones mostró una calidad global mucho mejor que el resto si se observan los valores respectivos en las columnas Media, DesvEstand y Peor. De cualquier manera, la opción sobre activas solamente muestra los mejores valores y una fuerte disminución del número de evaluaciones respecto de

la búsqueda sobre las restricciones 1, 2, 3, y 4 respectivamente. Este es el único caso para el cual fue necesario asignar un factor de penalización dinámico para obtener mejores resultados. Dicho factor de penalización es incrementado en cada iteración, lo que lleva a intensificar la búsqueda rápidamente alcanzando valores muy altos al final de la ejecución.

Problema G11

Tabla 6.10: Para G11 no es necesario usar penalización dado que contiene una única restricción por igualdad.

Restricción	Función G11 (0.75)					
	Mejor	Media	DesvEstand	Peor	# Factibles	Prom(#E)
Unica	0.75	0.75	0	0.75	30	70400

El problema G11, el cual está sujeto a sólo una restricción de igualdad no presentó ninguna dificultad para ser resuelto por SH_C , lo cual se puede observar (tabla 6.10) en la calidad de los resultados como así también en el número de soluciones factibles en las 30 corridas, es decir el 100 % de las mismas.

Tabla 6.11: Valor de penalización usado para G13 es $\mu(t) = 0,2$ para $t = 0, 1, \dots, t_{max}$.

Restricción	Función G13 (0.053950)					
	Mejor	Media	DesvEstand	Peor	# Factibles	Prom(#E)
1	0.053950	0.054908	0.00054	0.055386	6	29800
2	0.053950	0.054372	0.00044	0.054968	4	7400
3	0.053950	0.054637	0.00017	0.054394	6	7200
Activas	0.053950	0.054736	0.001	0.058462	15	19800

Problema G13

Similar a G11, el problema G13 sólo tiene restricciones de igualdad, aunque en este caso son 3. En la tabla 6.11 se puede observar que el algoritmo fue capaz de encontrar el mejor valor (muy cercano, por una diferencia de 0.000002), mostrando en todos los casos un desempeño similar, aunque la entrada “Activas” muestra una mayor cantidad de soluciones factibles. Cabe aclarar que la entrada “Activas” es equivalente a “Todas” para los casos en que el problema sólo contenga restricciones de igualdad como lo es el caso de este problema (G13).

Tabla 6.12: Valor de penalización usado para G14 es $\mu(t) = 100$ para $t = 0, 1, \dots, t_{max}$.

Restricción	Función G14 (50.80)					
	Mejor	Media	DesvEstand	Peor	# Factibles	Prom(#E)
1	-5.5080	-5.5080	0.0	-5.5080	30	5800
2	-5.5080	-5.5080	0.0	-5.5080	30	24000
Activas	-5.5080	-5.5080	0.0	-5.5080	30	21400

Tabla 6.13: Para G15, G16 y G17 no es necesario usar penalización dado que contienen una única restricción por igualdad.

Restricción	Mejor	Media	DesvEstand	Peor	# Factibles	Prom(#E)
Función G15 (16.73889)						
Única	-16.73889	-16.73889	0.0	-16.73889	30	10600
Función G16 (99.999)						
Única	99.999	99.999	0.0	99.999	30	32000
Función G17 (99.999)						
Única	99.999	99.999	0.0	99.999	30	11400

Problema G14

El análisis realizado para el problema G13 puede ser extendido al problema G14, el cual consta de dos restricciones y ambas activas en el óptimo. Vemos en la tabla 6.12 que en todos los casos, el algoritmo encontró la mejor solución y que a diferencia de G13, todas las soluciones encontradas, usando un número muy reducido de evaluaciones, fueron factibles. Esto se verifica particularmente cuando el algoritmo realiza la búsqueda sobre la restricción 1.

Problemas G14, G16 y G17

Vemos que para el problema G15, como para G16 y G17 en las tablas siguientes, el comportamiento del algoritmo es idéntico al caso G11 (calidad de las soluciones y porcentaje de soluciones encontradas en las 30 corridas), cuyo denominador común es que estos problemas tienen una única restricción por igualdad. Claramente se ve la eficacia el método en aquellos problemas que tienen una sola restricción del tipo $h(x)$.

6.2.1. Análisis global de SH_C

SH_C fue aplicado a 15 problemas numéricos con restricciones, variando en cantidad, tipo y dificultad. De los 15 problemas considerados, 11 de ellos han sido usados como *benchmarks* en muchos trabajos relacionados con la aplicación de nuevos enfoques para resolver problemas con restricciones. Los resultados expuestos aquí muestran que no sólo el enfoque propuesto es capaz de resolver dichos problemas, sino que también puede llegar a ser competitivo con respecto a otros algoritmos existentes. Cabe mencionar que la factibilidad de la aplicación del enfoque de frontera está supeditado a la existencia de soluciones óptimas sobre la frontera de al menos una de las restricciones del problema. Caso contrario, el enfoque podría ser desechado. Sin embargo, en muchos problemas numéricos restringidos se verifica que las soluciones óptimas están sobre la frontera. Otra característica que puede hacer factible su aplicación, es cuando el problema está sujeto solamente a restricciones del tipo $h(x)$. En esta situación una solución óptima necesariamente se encontrará sobre la frontera. Mas aun, cuando el problema está sujeto a sólo una restricción $h(x)$, en este caso puede aun ser más efectiva su aplicación dado que no es necesario introducir alguna técnica de manejo de restricciones.

Otra característica relevante del enfoque de frontera es la posibilidad de aplicación del método sin preocuparse por la eventual complejidad de la ecuación que define cada una de las restricciones del problema. La técnica descrita en la sección 5.4 mostró ser efectiva en todos los problemas considerados. A través de la misma se puede llegar tan cerca como se quiera a la frontera de las restricciones tomando un punto factible y uno no factible y sin necesidad de definir operadores especializados que preserven la factibilidad de la solución sobre la frontera. El caso más conocido de uso de operadores especializados para búsqueda en la frontera y que fue la motivación para el desarrollo de ciertos aspectos de esta tesis es el trabajo de Schoenauer et al. [157].

En la sección anterior se expusieron los resultados en forma individual para cada problema. Se realizó una comparación con los valores óptimos o mejores conocidos para cada uno de dichos problemas. En lo que sigue se realiza una breve comparación con dos métodos representativos de la literatura de AEs para la resolución algunos de los problemas considerados aquí. La tabla 6.14 muestra los valores obtenidos por SH_D ² y aquellos reportados en los artículos de Runnarson & Yao [151] (llamado RY) y uno más reciente de Hamida & Schoenauer [83] (llamado HS³). Para aquellos problemas que no fueron considerados por RY y/o HS se usa ND (No Disponible). La buena calidad de los resultados de SH_C no sólo se evidencia desde la perspectiva del Mejor Valor (primera columna), sino también observando los valores medios y peores para cada problema. De este grupo de problemas, hay un subconjunto que en general resulta muy fácil de resolver para la mayoría de los algoritmos encontrados en la literatura: G01, G03, G04, y G11. Los casos más interesantes de los restantes problemas incluyen a G02, G10 y G13. Estos problemas muestran en general un mayor grado de dificultad que se verifica en otros algoritmos propuestos para manejo de restricciones. El problema G02 fue resuelto por SH_C hasta el valor óptimo⁴ sin dificultad alguna y alcanzando valores muy similares a través de las 30 ejecuciones (ver Medio y Peor). Esta observación se sigue manteniendo para G13 respecto del Mejor, Medio y Peor valor. Sobre éste y otros problemas, se realizará en un capítulo posterior un análisis exhaustivo del desempeño de SH_C en relación a la velocidad de convergencia de dicho algoritmo. Por último, el problema G10 es el que en general ofrece un mayor grado de dificultad. Los resultados mostrados por SH_C son comparables a los de RY y HS. Se observa además, como característica relevante de SH_C para G10, una marcada disminución de los valores medios y peores en comparación con los otros dos algoritmos.

²Se toman los resultados encontrados usando la opción “Activas”.

³Mas conocido como ASCHEA.

⁴o mejor conocido, según corresponda.

Tabla 6.14: Comparación SH_C con los resultados de RY y HS

Prob.	Opt ^b	Mejor Valor			Valor Medio			Peor Valor		
		SH_C	RY	HS	SH_C	RY	KM	SH_C	RY	HS
G01	-15.000	-15.000	-15.000	-15.000	-14.786	-15.000	-14.84	-15.000	-15.000	ND
G02	0.803619	0.803619	0.803515	0.803614	0.802656	0.781975	0.788949	0.793083	0.726288	ND
G03	1.000	1.000	1.000	1	1.000	1.000	0.99997	1.000	1.000	ND
G04	-30665.539	-30665.539	-30665.539	-30665.5	-30665.539	-30665.539	-30665.5	-30666.539	-30665.539	ND
G05	5126.498	5126.50	5126.497	5126.5	5138.37	5128.881	5126.53	5132.14	5142.472	ND
G06	-6961.814	-6961.81	-6981.814	-6961.81	-6961.74	-6875.940	-6961.81	-6961.71	-6350.262	ND
G07	24.306	24.37	24.307	24.3323	24.64	24.374	24.6636	24.92	24.642	ND
G09	680.630	680.63	680.63	680.63	680.67	680.56	680.641	680.72	680.763	ND
G10	7049.331	7052.30	7054.316	7049.33	7199.01	7559.192	7212.35	7943.15	8835.655	ND
G11	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75	ND
G13	0.053950	0.053950	0.053957	ND	0.054908	0.057006	ND	0.055386	0.216915	ND
G14	-5.5080	-5.5080	ND	ND	-5.5080	ND	ND	-5.5080	ND	ND
G15	-16.73819	-16.73819	ND	ND	-16.73819	ND	ND	-16.73819	ND	ND
G16	99.999	99.999	ND	ND	99.999	ND	ND	99.999	ND	ND
G17	99.999	99.999	ND	ND	99.999	ND	ND	99.999	ND	ND

6.3. Problemas discretos (aplicación de SH_D)

Esta sección comienza describiendo para cada problema discreto, aquellas componentes del algoritmo SH_D que dependen del problema al cual es aplicado. Es importante hacer notar que dada la robustez de este enfoque en general y para problemas de subconjunto en particular, la modificación fundamental a ser realizada es sobre la función encargada de calcular los valores de la heurística η .

Esta sección presenta en primer término la descripción de la obtención de los valores heurísticos para los problemas MKP, SCP y MISP respectivamente, acompañadas con un ejemplo concreto en cada caso. A continuación se presentan por separado los resultados experimentales sobre un conjunto de instancias. La sección finaliza con un análisis y consideraciones generales acerca de SH_D y su desempeño sobre los problemas e instancias consideradas.

Independientemente del problema considerado, los valores de la heurística $\eta_i(t)$ para cada componente factible i respecto de la hormiga k en el tiempo t y el rastro asociado correspondiente, pueden ser visualizados como un conjunto $\mathcal{A}_k = \{(j_1, \eta_{j_1}, \tau_{j_1}), \dots, (j_s, \eta_{j_s}, \tau_{j_s})\}$, con $s \leq n$, siendo n el tamaño del problema o número total de componentes del problema bajo consideración. En consecuencia, en cada paso de la construcción de una solución, una hormiga k tiene asociado un conjunto de componentes elegibles (por ser factibles) el cual es modificado cada vez que una

nueva componentes es incluida en la solución. La modificación incluye en principio, a) la eliminación de \mathcal{A}_k de la componentes recientemente incorporadas, y b) la eliminación de todas aquellas componentes que se convierten en no factibles en ese momento, según las restricciones del problema. Claramente, el principal cambio a tener en cuenta está vinculado con la heurística específica y es el objetivo del presente capítulo, es decir desarrollar y ejemplificar las heurísticas propuestas para cada uno de los problemas estudiados. A continuación se proponen algunas heurísticas muy simples para MKP, SCP y MISP.

6.3.1. SH_D para MKP

Con el objeto de definir la heurística local para MKP, es necesario considerar una instancia en particular de este problema, por ejemplo aquella dada en la sección 5.7.2.

Cabe recordar que $\tilde{S}^k(t)$ denota el conjunto de componentes que han sido seleccionadas por la hormiga k en el tiempo t (ver sección 5.4.2). Supongamos que $\tilde{S}^k(1) = \{4\}$. Antes de dar la expresión para la heurística local, es necesario dar algunas definiciones previas.

Sea $u_i(k, t) = \sum_{l \in \tilde{S}^k(t)} r_{il}$ la cantidad de recursos i consumidos en el tiempo t con respecto a la solución que está siendo construida por la hormiga k . Siguiendo el ejemplo, tenemos:

$$\begin{aligned} u_1(k, t) &= \sum_{l \in \{4\}} r_{1l} = r_{14} = 1 \\ u_2(k, t) &= \sum_{l \in \{4\}} r_{2l} = r_{24} = 3 \\ u_3(k, t) &= \sum_{l \in \{4\}} r_{3l} = r_{34} = 2 \end{aligned}$$

Sea $\gamma_i(k, t) = c_i - u_i(k, t)$ la cantidad remanente para alcanzar el límite de la restricción i , es decir, lo que queda del recurso i .

$$\begin{aligned} \gamma_1(k, t) &= c_1 - u_1(k, t) = 8 - 1 = 7 \\ \gamma_2(k, t) &= c_2 - u_2(k, t) = 12 - 3 = 9 \\ \gamma_3(k, t) &= c_3 - u_3(k, t) = 10 - 2 = 8 \end{aligned}$$

La siguiente fórmula da una indicación de cuán ajustada está la componente j en relación a la solución $\tilde{S}^k(t)$, es decir, la razón entre r_{ij} , la cantidad del recurso i consumido por el proyecto j , y $\gamma_i(k, t)$. De esta manera, mientras más pequeña sea dicha razón, mayor debería ser la preferencia por la componente j ,

$$\delta_{ij}(k, t) = \frac{r_{ij}}{\gamma_i(k, t)} \quad (6.2)$$

Luego, cada $\delta_{ij}(k, t)$ es obtenido como sigue:

$$\delta_{11}(k, t) = \frac{4}{7} \quad \delta_{12}(k, t) = \frac{4}{7} \quad \delta_{13}(k, t) = \frac{2}{7}$$

$$\delta_{21}(k, t) = \frac{6}{9} \quad \delta_{22}(k, t) = \frac{6}{9} \quad \delta_{23}(k, t) = \frac{3}{9}$$

$$\delta_{31}(k, t) = \frac{4}{8} \quad \delta_{32}(k, t) = \frac{4}{8} \quad \delta_{33}(k, t) = \frac{2}{8}$$

Finalmente, definimos el ajuste promedio sobre todas las restricciones i , asumiendo que la componente j es la próxima candidata para ser incluida en $\tilde{S}^k(t)$, como:

$$\bar{\delta}_j(k, t) = \frac{\sum_{i=1}^m \delta_{ij}(k, t)}{m} \quad (6.3)$$

Para nuestro ejemplo, los valores mayores son aquellos obtenidos para las componentes $j = 1$ y $j = 2$:

$$\bar{\delta}_1(k, t) = 0,579; \quad \bar{\delta}_2(k, t) = 0,579; \quad \bar{\delta}_3(k, t) = 0,290$$

Sin embargo, es necesario tener en cuenta los beneficios p_j a fin de obtener una medida de *pseudo-utilidad* para cada componente candidata. Por lo tanto, definimos la heurística local para MKP, $\eta_j(\tilde{S}_k(t))$, de la siguiente manera:

$$\eta_j(\tilde{S}_k(t)) = \frac{p_j}{\bar{\delta}_j(k, t)} \quad (6.4)$$

Siguiendo con el ejemplo, la componente que acreditaría mayor beneficio es la $j = 2$, dado que su beneficio p_2 es mayor que p_1 . De esta manera, los valores de η_j para $j = 1, 2, 3$ son:

$$\begin{aligned}\eta_1(\{4\}) &= \frac{4}{0,579} = 6,908, \quad \eta_2(\{4\}) = \frac{10}{0,579} = 17,271, \\ \eta_3(\{4\}) &= \frac{2}{0,290} = 6,897;\end{aligned}$$

las cuales influirán en los valores de probabilidad para la selección de una determinada componente (ecuación 5.6). Por consiguiente, los valores de probabilidad representan un balance entre la *pseudo-utilidad*, la cual dice que las componentes preferibles son aquellas que usan menor cantidad de recursos, y la intensidad del rastro, la cual dice que si una componente j está incluida en muchas soluciones, ésta es preferible.

Una estructura de datos llamada *lista tabú* es asociada a cada hormiga a fin de prevenir que las mismas elijan una componente más de una vez, es decir que, $tabu_k(t)$ mantiene el conjunto de componentes que son parte de la solución que está en proceso de construcción en el tiempo t por la hormiga k . La lista tabú mantiene además, los valores $u_j(k, t)$ ($j = 1, \dots, m$) con el objeto de reducir los tiempos computacionales consumidos en el cálculo de los valores heurísticos.

Luego, el conjunto $\mathcal{N}_k(t)$ ⁶ puede ser definido como:

$$\mathcal{N}_k(t) = \{j | j \notin tabu_k(t) \text{ y la solución } \tilde{S}^k(t) \text{ satisface todas las restricciones, si la componente } j \text{ es agregada a dicha solución}\},$$

y la lista $tabu_k(t)$ representa $\tilde{S}^k(t)$ según la definición dada en la sección 5.4.2. Dado que MKP es un problema de maximización, luego $G(L_k) = QL_k$, donde $Q = \frac{1}{\sum_{j=1}^n p_j}$ ⁷.

⁶Esta notación corresponde a la usada en la descripción de la metaheurística ACO (sección 3.1.2).

⁷ Q es un parámetro explicado en la sección 3.1.3.

6.3.2. SH_D para SCP

La heurística local (basada en un operador de reparación descrito en [32]) es definida de la siguiente manera:

$$\eta_j(\tilde{S}_k(t)) = \frac{|U_k(t) \cap CR_j|}{c_j}, \quad (6.5)$$

donde $U_k(t)$ es el conjunto de filas aún no cubiertas por la hormiga k en el tiempo t , CR_j es el conjunto de filas cubiertas por la columna j y c_j es el costo de la columna j . De esta manera, mientras más alto sea el valor de dicha razón, mayor preferencia se debería tener por la columna j . La probabilidad de selección de una componente está dada por la ecuación 5.6, donde $\mathcal{N}_k(t) = \{j/j \notin \text{tabu}_k(t)\}$ y $\text{tabu}_k(t)$ es $\tilde{S}_k(t)$ según la definición dada en la sección 5.4.2. La función G , la cual determina la cantidad de rastro de feromona (ver ecuación 5.5) de acuerdo al valor de la función objetivo $L_k = \sum_{j=1}^n c_j x_j$, se define como sigue:

$$G(L_k) = \frac{\sum_{j=1}^n c_j}{L_k}.$$

El ejemplo dado en la sección 5.7.2 para SCP es usado para intentar visualizar cómo los valores más altos para la heurística reflejan el hecho que una columna se torna preferible para ser incluida como parte de la solución. Supongamos que en el tiempo t la solución parcial en proceso de contrucción por la hormiga k es $\tilde{S}^k(t) = \{2\}$. Así, el conjunto de filas cubiertas es $\{1, 5\}$; $U_k(t) = \{2, 3, 4, 6\}$ y el conjunto de componentes factibles (no redundantes) que pueden ser seleccionadas es $\{1, 3, 4, 5, 6, 7, 8, 9, 10\}$. Luego, el respectivo valor de la heurística para cada uno de ellos es:

$$\eta_1(\{2\}) = \frac{|\{2,3,4,6\} \cap \{3,5\}|}{c_1} = \frac{|\{3\}|}{5} = \frac{1}{5} = 0,2$$

$$\eta_3(\{2\}) = \frac{|\{2,3,4,6\} \cap \{5,6\}|}{c_3} = \frac{|\{6\}|}{3} = \frac{1}{3} = 0,333$$

$$\eta_4(\{2\}) = \frac{|\{2,3,4,6\} \cap \{3\}|}{c_3} = \frac{|\{3\}|}{3} = \frac{1}{2} = 0,5$$

$$\eta_5(\{2\}) = \frac{|\{2,3,4,6\} \cap \{1,4\}|}{c_5} = \frac{|\{4\}|}{8} = \frac{1}{8} = ,125$$

$$\eta_6(\{2\}) = \frac{|\{2,3,4,6\} \cap \{6\}|}{c_6} = \frac{|\{6\}|}{4} = \frac{1}{4} = 0,25$$

$$\eta_7(\{2\}) = \frac{|\{2,3,4,6\} \cap \{1,2,4,5\}|}{c_7} = \frac{|\{2,4\}|}{4} = \frac{2}{4} = ,5$$

$$\eta_8(\{2\}) = \frac{|\{2,3,4,6\} \cap \{2\}|}{c_8} = \frac{|\{2\}|}{3} = \frac{1}{3} = 0,333$$

$$\eta_9(\{2\}) = \frac{|\{2,3,4,6\} \cap \{1,6\}|}{c_9} = \frac{|\{6\}|}{5} = \frac{1}{5} = 0,2$$

$$\eta_{10}(\{2\}) = \frac{|\{2,3,4,6\} \cap \{3,5\}|}{c_{10}} = \frac{|\{3\}|}{2} = \frac{1}{2} = 0,5$$

Según las columnas factibles remanentes, podemos observar que las columnas 4, 7, y 10 son las mejores candidatas. Esto es porque ellas tienen asociado el mayor valor de η con respecto a las filas aún no cubiertas y sus respectivos costos de inclusión.

6.3.3. SH_D para MISP

La heurística local para MISP es definida considerando por adelantado el número de vértices factibles disponibles en el caso que uno de ellos sea seleccionado para ser incluido en la solución en proceso de construcción. Sea $F_k(t)$ el conjunto de componentes factibles con respecto a $\tilde{S}_k(t)$, la solución que está siendo construida por la hormiga k en el tiempo t . Por lo tanto, la heurística local MISP es definida como:

$$\eta_i(\tilde{S}_k(t)) = |F_i|, \quad (6.6)$$

donde F_i representa $F_k(t+1)$ en caso de que la componente i sea añadida a \tilde{S}_k .

De esta manera, la heurística local intenta asignar un puntaje más alto a aquellas columnas (digamos i) que mantienen el conjunto F_i con la mayor cantidad posible de elementos. Así, mientras más grande sea el conjunto $F_k(t+1)$, más grande será el conjunto de elementos remanentes para completar la solución parcial \tilde{S}_k .

La probabilidad para seleccionar una componente en particular está dada por la ecuación 5.6 donde $\mathcal{N}_k(t) = V - \tilde{S}_k(t) - U_k(t)$ y $U_k(t) = \{j / (j, i) \in E \vee (i, j) \in E \wedge i \in \tilde{S}_k(t)\}$; v.g., el conjunto de componentes factibles con respecto a $\tilde{S}_k(t)$. La función G está definida como $G(L_k) = QL_k$ donde $Q = 1/|V|$ y L_k , el valor de la función objetivo, es la cardinalidad del conjunto de vértices que conforman la solución contruida por la hormiga k .

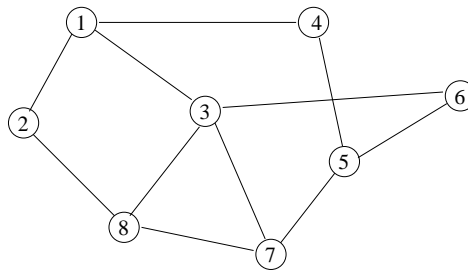


Figura 6.3: Instancia de MISP

Consideremos para este problema la instancia de la figura 6.3 (ya dada en la sección 5.7.2) a fin de mostrar los correspondientes valores de la heurística. Supongamos que en el tiempo t la solución parcial que está siendo cosntruida por la hormiga k es $\tilde{S}^k(t) = \{2\}$, luego $F_k(t) = V - \{2\} - \{3, 4\} = \{1, 5, 6, 7, 8, 9, 10\}$. En la última expresión, el conjunto $\{3, 4\}$ representa el subconjunto de componentes factibles debido a la inclusión de la componente 2 en la solución parcial. Ahora el subconjunto $\{1, 5, 6, 7, 8, 9, 10\}$ es el conjunto de componentes factibles y los correspondientes valores de la heurística son:

$$\begin{aligned}
\tau_1(S^k(t)) &= |F_1| = |\{10\}| = 1 \\
\tau_5(S^k(t)) &= |F_5| = |\{8\}| = 1 \\
\tau_6(S^k(t)) &= |F_6| = |\{3, 7, 9\}| = 3 \\
\tau_7(S^k(t)) &= |F_7| = |\{6, 8\}| = 2 \\
\tau_8(S^k(t)) &= |F_8| = |\{5, 7\}| = 2 \\
\tau_9(S^k(t)) &= |F_9| = |\{6\}| = 1 \\
\tau_{10}(S^k(t)) &= |F_{10}| = |\{1\}| = 1
\end{aligned}$$

Por lo tanto, el más alto puntaje es obtenido por la componente $i = 6$ la cual es la componente que deja el mayor conjunto de componentes para el próximo paso de selección en el proceso de construcción.

6.3.4. Experimentos y resultados

De los tres problemas considerados en la presente tesis, el más importante desde el punto de vista de la profundidad del estudio realizado es MISP, luego le sigue SCP y por último MKP. Sin embargo, la aplicación de SH_D comenzó por MKP (el reporte más importante para este problema puede ser encontrado en [106, 49]), luego SCP⁸ y por último MISP [66, 49]. En este mismo orden serán presentados en esta sección. Cabe aclarar que algunas tablas de resultados, debido a su tamaño, son incluídas al final del capítulo.

Multiple Knapsack Problem

Los estudios preliminares de SH_D aplicado a MKP involucró la variación de parámetros claves de dicho algoritmo. Estos parámetros fueron α y β , los cuales controlan respectivamente la importancia de la información global (rastros) y local (valor heurístico).

⁸Para este problema no existen publicaciones previas de los resultados, aunque se menciona la aplicación de este enfoque para SCP en Dorigo et al. [49].

Los resultados reportados para MKP incluye un estudio inicial sobre un conjunto de instancias de MKP obtenidas de la OR Library [20] a fin de ajustar los parámetros de SH_D referidos en el párrafo anterior (tablas 6.15 y 6.16). Estas instancias las denominaremos $for1^*$ y $for2^*$ según aparecen nombradas en [20].

El otro experimento reportado, incluye un subconjunto de instancias generadas por Paul Chu [32] como parte del estudio de su tesis doctoral. Dichas instancias también están disponibles en la OR Library [20]. En su tesis, Chu propone, como parte de su estudio, un nuevo conjunto de instancias de MKP dado que las instancias existentes hasta ese momento para ser usadas como *benchmarks*, resultaron ser demasiado simples debido a dos factores principales: el tamaño y/o el grado de restricción del espacio de búsqueda. Estas nuevas instancias (parte de ellas son mostradas en la tabla 6.17) fueron construidas con un grado de dificultad mayor dado principalmente por la capacidad de las mochilas (valores de r_j).

Para los experimentos reportados en esta sección, fueron considerados los siguientes valores de parámetros : $\alpha = 1, 5$; $\beta = 1, 5, 9$; $\rho = 0, 5$, número de hormigas a igual a n , donde $n = |S|$, es decir, la cardinalidad de MKP. El número máximo de ciclos o iteraciones fue establecido en 100 para todos los experimentos. Los resultados están expresados en término del promedio de 30 corridas con diferentes semillas para el generador de números aleatorios. El vector que representa el rastro de feromona es inicializado en forma aleatoria, donde $\tau_i \in (0, 1)$, $i \in \{1, \dots, n\}$.

SH_D fue probado inicialmente sobre dos grupos de 18 instancias en total del tipo $for1^*$ y $for2^*$ mencionadas previamente. Las del primer grupo son instancias pequeñas y no muy difíciles. SH_D encontró para cada una de ellas los óptimos en todas las corridas en las primeras generaciones y para la mayoría de las combinaciones de parámetros, lo cual permitió poder realizar un análisis claro de la influencia de los valores de los parámetros de interés. Por esta razón sólo se muestran los resulta-

dos preliminares para las instancias $for2^*$ realizando una variación sobre los parámetros α y β . Las tablas 6.15 y 6.16 muestran, para cada instancia, el respectivo valor óptimo, el promedio sobre los mejores valores encontrados y el número de veces de las 30 corridas que el algoritmo encontró una solución óptima.

Tabla 6.15: Resultados para 11 instancias de MKP

		(α, β)					
Instance	Opt	(1,1)		(1,5)		(1,9)	
for2-1	7772	7772	30	7772	20	7772	30
for2-2	8722	8717	12	8722	20	8719	18
for2-3	141278	141078	18	140778	0	140778	0
for2-4	130883	130645	15	130819	18	130883	30
for2-5	95677	95667	24	95667	24	95667	12
for2-6	119337	119337	30	119337	30	119337	30
for2-7	98796	98796	30	98796	30	98796	30
for2-8	130623	130389	12	130623	30	130311	3
for2-9	1095445	1095253	0	1095382	0	1095382	0
for2-10	624319	622821	15	624319	30	622238	3
for2-11	4554	4554	30	4554	30	4554	30

Tabla 6.16: Resultados para 11 instancias de MKP (continuación)

		(α, β)					
Instance	Opt	(5,1)		(5,5)		(5,9)	
for2-1	7772	7635	0	7768	21	7770	24
for2-2	8722	8655	0	8720	21	8716	0
for2-3	141278	139583	0	140778	0	140778	0
for2-4	130883	127558	0	130439	0	130787	12
for2-5	95677	94351	0	95657	18	95637	3
for2-6	119337	116690	3	119337	30	119337	30
for2-7	98796	97460	12	98796	30	98796	30
for2-8	130623	125742	0	130311	6	130233	0
for2-9	1095445	1092659	0	1095376	0	1095382	0
for2-10	624319	615414	0	624178	24	622066	0
for2-11	4554	4491	0	4554	30	4554	30

Los resultados reportados en las tablas 6.15 y 6.16 indican que el mejor desempeño (en este caso, considerando el mayor número de veces que se encontró el óptimo) es obtenido en los casos donde $\alpha = 1$. Se hace notar que en la instancia for2-9 el algoritmo falló en encontrar el valor óptimo para todas las combinaciones de parámetros consideradas. Para esta instancia, el mejor resultado obtenido es igual al valor sub-óptimo (1095382) reportado por Khuri et al. [12]. También es importante notar

que las instancias de mayor tamaño para este conjunto de instancias corresponden a las instancias for2-9 y for2-10 las cuales tienen $n = 105$ variables y $m = 5$ restricciones; el resto de las instancias son de menor tamaño.

El otro grupo de instancias consideradas, según se mencionó previamente, también fueron obtenidas de [20]. Para estas instancias, fue usada la combinación de parámetros $(\alpha, \beta) = (1, 5)$, que fue una de las combinaciones que mostró el mejor comportamiento del algoritmo para las instancias consideradas en los experimentos preliminares. Este conjunto de instancias tienen $n = 100$ variables y $m = 5$ o $m = 10$ restricciones⁹. Las columnas de la tabla 6.17 muestra el Mejor Valor Conocido¹⁰ (MVC); el Mejor Valor Encontrado (MVE), el promedio respectivo de 30 corridas (Prom(MVE)) y el número promedio de evaluaciones necesarias para alcanzar los mejores valores (Prom(#E)).

Puede observarse que SH_D encontró soluciones con valores objetivos iguales a MVC para 13 de las 20 instancias consideradas con 100 variables. Para estas 13 instancias, la mejor solución fue encontrada dentro de las primeras 4000 evaluaciones, excepto para 3 de ellas, para las cuales el algoritmo necesitó más de 7000 evaluaciones.

Es importante destacar en este punto el muy buen desempeño de SH_D con el conjunto de instancias probadas siendo que éste es un algoritmo de propósito general para este tipo de problemas y que sólo incluye conocimiento del problema a través de la heurística. Esto implica que existe más de una posibilidad de incorporar algún tipo de las denominadas *acciones auxiliares* (sección 3.1.2) como por ejemplo, realizar búsqueda local a partir de la mejor solución encontrada y modificar el rastro de feromona según la nueva solución encontrada si ésta es mejor que la original. Esta forma híbrida de las metaheurísticas es uno de los campos de investigación más prolíficos en la actualidad.

⁹El nombre m - n - q de una instancia de este tipo indica: m restricciones, n variables, y el número de secuencia q .

¹⁰Estos valores fueron reportados por primera vez en el trabajo de P.Chu [32].

Tabla 6.17: Resultados de SH_D para instancias más difíciles de MKP

Instancia	MVC	MVE	Prom(MVE)	Prom(#E)
5.100-00	24381	24381	24331.2	3500
5.100-01	24274	24274	24245.6	2300
5.100-02	23551	23551	23527.6	1100
5.100-03	23534	23527	23463.0	7800
5.100-04	23991	23991	23949.8	3400
5.100-05	24613	24613	24563.0	2200
5.100-06	25591	25591	25504.8	3500
5.100-07	23410	23410	23361.8	2200
5.100-08	24216	24204	24173.4	4300
5.100-09	24411	24411	24326.0	1700
10.100-00	23064	23057	22996.4	5900
10.100-01	22801	22801	22672.2	5500
10.100-02	22131	22131	21980.0	2400
10.100-03	22772	22772	22631.0	7200
10.100-04	22751	22654	22578.4	4200
10.100-05	22777	22652	22565.2	7700
10.100-06	21875	21875	21758.2	2100
10.100-07	22635	22551	22519.4	1100
10.100-08	22511	22418	22292.4	6200
10.100-09	22702	22702	22588.0	2400

Set Covering Problem

Los valores de los parámetros de SH_D para el primer conjunto de experimentos que son reportados en esta sección, son similares a los usados para MKP, excepto por el número de hormigas. Para MKP, el número de hormigas en la colonia era equivalente al tamaño de la instancia, sin embargo, considerar el mismo criterio para las instancias de SCP llevaría a un excesivo gasto computacional. Las instancias de SCP consideradas fueron obtenidas de la OR Library [20], y son mostradas en las tablas 6.18 y 6.19. Como puede observarse, el número de columnas en general muy alto en la mayoría de los casos. El número de filas, otro factor importante relacionado con el gasto computacional, afecta directamente el tiempo necesario para construir las soluciones y en consecuencia, un mayor número de filas implica también un alto gasto computacional. Es importante aclarar que los algoritmos ACO y EA presentados en esta tesis usaron una versión pre-procesada de las instancias de la tabla 6.18. El método usado es el mismo que el aplicado por Marchiori et al. [110] en su estudio computacional para SCP. Lo importante a destacar es que

luego del pre-procesamiento, los respectivos tamaños de las instancias (número de columnas solamente) se redujeron en forma muy significativa en la mayoría de los casos (entre un 50 % y 80 %). Las instancias de la tabla 6.19 también fueron pre-procesadas, pero no fue posible reducir su tamaño para conseguir un problema equivalente que implique menos gasto computacional.

Las principales características de estas instancias es que cada columna cubre al menos una fila y cada fila es cubierta por al menos dos columnas. La columna *Densidad* representa la fracción de 1s con respecto al número total de componentes de la matriz.

Con respecto al vector que representa el rastro de feromona, como en los anteriores experimentos para MKP, es inicializado en forma aleatoria de manera tal que $\tau_i \in (0, 1)$, $i \in \{1, \dots, n\}$.

Tabla 6.18: Instancias de SCP consideradas en el estudio (primer grupo)

Tipo de Instancia	Filas	Columnas	Densidad (%)	Número de Instancias
4	200	1000	2	5
6	200	1000	5	5
A	300	3000	2	5
B	300	3000	5	5
C	400	4000	2	5
D	400	4000	5	5
E	50	500	20	5
NRE	500	5000	10	5
NRF	500	5000	20	5
NRG	1000	10000	2	5
NRH	1000	10000	5	5

Los primeros resultados (sobre las instancias de tipo 4 y 6) para algunas combinaciones de los parámetros α y β son mostrados en la tabla 6.20 usando 50 hormigas en la colonia. Los valores en cada columna representan los mejores resultados de 30 corridas y el número de ciclos necesarios para obtener dicho resultado. Claramente, SH_D no es capaz de obtener

Tabla 6.19: Instancias adicionales de SCP (segundo grupo)

Instancia	Filass	Columnas	Densidad (%)
CYC.6	240	192	2.1
CYC.7	672	448	0.9
CYC.8	1792	1024	0.4
CYC.9	4608	2304	0.2
CLR.10-4	511	210	12.3
CLR.11-4	1023	330	12.4
CLR.12-4	2047	495	12.5
CLR.13-4	4095	715	12.5

los mejores valores conocidos para dichas instancias, sin embargo, la performance alcanzada es bastante razonable teniendo en cuenta la cercanía de los valores obtenidos a los óptimos de cada una de dichas instancias.

Tabla 6.20: Resultados preliminares obtenidos para algunas instancias de SCP

Instancia	Opt. Conocido	(α, β)					
		(1,5)		(5,5)		(5,9)	
		Sol	#ciclos	Sol	#ciclos	Sol	#ciclos
4.1	429	447	7	444	5	445	6
4.2	512	554	6	547	5	553	6
4.3	516	556	6	535	8	546	5
4.4	494	520	21	509	32	518	14
4.5	512	538	28	518	16	544	17
6.1	138	145	28	144	18	146	18
6.2	146	157	19	156	18	150	16
6.3	145	150	8	149	20	147	21
6.4	131	136	11	135	6	136	15
6.5	161	177	24	176	23	167	27

Con el objeto de mejorar la performance de SH_D , se incrementó el número de hormigas al doble, es decir una colonia de tamaño 100. Los resultados obtenidos para la combinación $(\alpha = 5, \beta = 9)$ son mostrados en la tabla 6.21. La calidad de los soluciones mejoró en forma significativa, llegando en casi todos los casos a valores muy cercanos a los óptimos,

aunque sólo para tres instancias el óptimo fue alcanzado, correspondiendo a las instancias 6.1, 6.3 y 6.4. Incrementos posteriores sobre el número de hormigas no llevaron a mejorar el comportamiento de este algoritmo. Otro punto a considerar es que el número de ciclos también se incrementó. Considerando esto y la mejora en la calidad de los resultados, se verifica un comportamiento más explorativo y menos explotativo, es decir, una disminución general en la velocidad de convergencia del algoritmo.

Tabla 6.21: Resultados para las instancias de SCP tipo 4* y 6* con un incremento en el número de hormigas.

Instancia	Opt. Conocido	$(\alpha, \beta) = (5, 9)$	
		Sol	#ciclos
4.1	429	429	46
4.2	512	514	30
4.3	516	520	51
4.4	494	500	65
4.5	512	515	35
6.1	138	139	29
6.2	146	150	39
6.3	145	145	60
6.4	131	131	41
6.5	161	165	31

Posteriores experimentos sobre el resto de las instancias de SCP fueron realizados considerando valores más altos para el parámetro β , es decir, se incrementó la importancia del valor heurístico usado en el proceso de selección de componentes en la etapa de construcción de una solución. De manera similar, se incrementó el número de ciclos a 500. En síntesis, los valores de parámetros considerados fueron los siguientes: $\alpha = 1$; $\beta = 2, 3, 10$; $\rho = 0,5$ y el número total de hormigas en la colonia igual a 100.

El grupo de instancias de la tabla 6.18 fueron usadas en el estudio de P. Chu [32]. De este grupo, las instancias del tipo 4*, 6*, A, B*, C*, D* y E*

tienen asociados valores óptimos probados a través de métodos exactos. El segundo grupo; las del tipo NRE*, NRF*, NRG* y NRH*, tiene asociados valores mejores conocidos (es decir, óptimos no conocidos). El AG propuesto por P.Chu representa uno de los algoritmos representante del estado del arte para estas instancias. Por un lado, las instancias del primer grupo fueron resueltas en su totalidad. Para las del segundo grupo, se dieron dos situaciones: se llegó en mucho casos a los valores mejores conocidos, mientras que en los restantes casos de mejoraron los mejores valores conocidos al momento. En consecuencia, hemos incluido en nuestro estudio comparativo de los resultados reportados por P. Chu [32] para SCP. Dichos valores conforman, al menos hasta el momento de la realización del presente estudio, un importante punto de referencia para un estudio comparativo. El grupo de instancias de la tabla 6.19 no fueron consideradas en el estudio de P. Chu, aunque aquí son incluidas y los valores considerados corresponden a valores óptimos reportados en la OR Library.

También se incluyó en la comparación un Algoritmo Evolutivo de propósito general llamado GENEsYs [7] el cual incluye rutinas para la resolución de SCP. GENEsYs no es un algoritmo competitivo para SCP, aunque es interesante su consideración como un ejemplo de métodos alternativos, en este caso, un EA que usa la técnica de reparación como técnica para manejo de restricciones. Los parámetros para GENEsYs fueron escogidos de manera tal de lograr la mejor calidad posible de los resultados. Algunos de ellos son: representación binaria, $\mu = 15$ (número de padres), $\lambda = 100$ hijos¹¹ y un número de generaciones igual a 1000. Se escogió crossover uniforme y mutación estándar como operadores genéticos. Ambos operadores están complementados con un proceso de reparación en caso que las soluciones resultantes no sean factibles.

La tabla 6.22 muestra los resultados obtenidos para las instancias de las tablas 6.18 y 6.19. Por cada una de ellas se muestra: el nombre de la

¹¹ Aquí se usa el método (μ, λ) descrito en la sección 2.5.7.2

instancia, el valor Óptimo o MVC según corresponda, luego para SH_D y $GENEsYs$ se muestra respectivamente el Mejor Valor Encontrado (MVE), el promedio de MVE ($Prom(MVE)$) y el número de veces que el algoritmo alcanzó el óptimo o MVC ($\#hits$)¹².

En primer término es importante destacar que el desempeño global de SH_D está situado entre los correspondientes a $GENEsYs$ y los valores obtenidos por P.Chu. $GENEsYs$ muestra un desempeño pobre en términos de $\#hits$; como puede observarse, en sólo una de las instancias logró alcanzar el óptimo. Por su parte, SH_D no logra alcanzar los valores óptimos o MVC en todos los casos, sin embargo la calidad de MVE y sus respectivos promedios, muestran un comportamiento muy satisfactorio de este algoritmo en el cual las soluciones generadas están sobre la frontera. Con respecto a la densidad de las instancias, vemos una marcada influencia en el desempeño de SH_D y $GENEsYs$. Además de lo que pueda observarse en la tabla 6.22, esta situación puede apreciarse en la figura 6.4 desde una perspectiva más general en donde se muestra gráficamente el promedio del error porcentual¹³ de los valores obtenidos para el conjunto total de instancias agrupadas según sus densidades.

Siguiendo con la descripción de la tabla 6.22, observamos lo siguiente. Por ejemplo, para las instancias con una densidad de 1s del 20 % (E^* y NRF^*) y 10 % (NRE^*), ambos algoritmos muestran su mejor desempeño, haciendo la diferencia en que SH_D alcanzó, en varias oportunidades, los óptimos o MVCs. A medida que la densidad disminuye (5 % para B^* y D^*) se aprecia un impacto negativo sobre la calidad de los resultados. Esta situación se aprecia aún más para las instancias A^* y C^* , ambos tipos con una densidad del 2 %. Esto no implica una generalización en relación a esta característica de las instancias. Por ejemplo, para NRG^* y NRG^* , el comportamiento es similar en relación a $\#hits$, pero se mantienen las anteriores diferencias si se observan los valores promedios respectivos.

¹²Usaremos el término en Inglés *hits* a través de este documento.

¹³($\%error = ((Opt - MVC)/Opt) * 100$) donde $n Opt$ indica el valor óptimo o mejor conocido de la instancia respectiva.

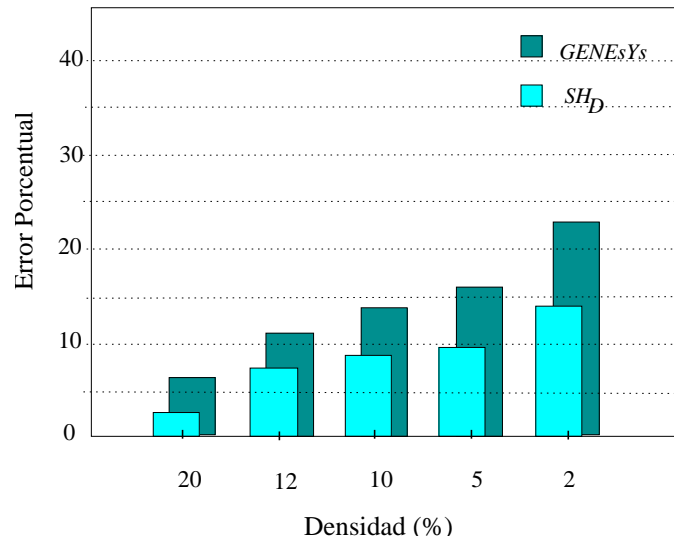


Figura 6.4: La figura muestra el promedio del error porcentual según Prom(MVE) para todas las instancias agrupadas por densidad.

En estos últimos casos, el tamaño de las instancias es un factor clave que, junto a la densidad, definen la complejidad de las instancias. Una situación similar se observa para las instancias CYC* y CLR*. Vemos que los mejores valores alcanzados corresponden a las instancias de mayor densidad (CLR* y CYC.6). Si bien para las CLR* la densidad de alta, existe otro factor que es el gran número de filas que contienen, lo que afecta también el desempeño de los algoritmos.

Los valores de los parámetros α y β no mostraron un comportamiento definitivo para las instancias consideradas. Cabe aclarar que los resultados mostrados en la tabla 6.22 para SH_D no corresponden a una única combinación de estos parámetros. Sin embargo, se puede concluir de las observaciones realizadas, que la mejor combinación corresponde a $(\alpha = 1, \beta = 2)$, y siguiendo en el orden, $(\alpha = 1, \beta = 3)$ y $(\alpha = 1, \beta = 10)$. Esto indica que en muchos casos, darle una importancia similar a la información global y local, permite al algoritmo encontrar los mejores valores. Sin embargo, esto no se verificó para todas las instancias. En algunos casos fue necesario incrementar fuertemente la importancia del

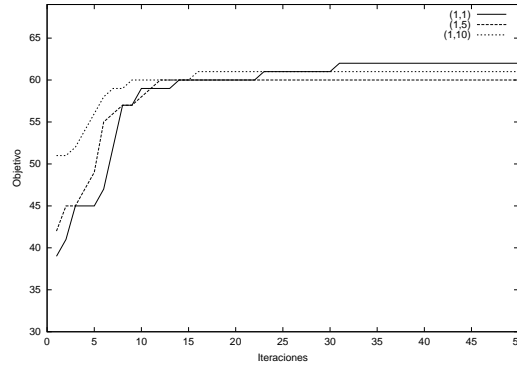


Figura 6.5: Curva de convergencia para distintos valores de β .

valor heurístico ($\beta = 10$) para conseguir soluciones de mejor calidad¹⁴. Asimismo, se destaca que SH_D se comportó de manera similar para las instancias de mayor densidad, para todas las combinaciones de los valores de α y β , aunque se verifica que se produce una aceleración en la convergencia para valores altos de β . Esta situación influye negativamente cuando la instancia considerada presenta un mayor grado de dificultad, esto es, el algoritmo se comporta en forma “más *greedy*” o bien, más explotativo que explorativo, lo que en general conduce a óptimos locales en el espacio de búsqueda respectivo.

Maximum Independent Set Problem

Los valores de parámetros usados con SH_D para MISP son iguales para todos los experimentos mostrados en esta sección. Los mismos son $\alpha = 1$, $\beta = 1$, $\rho = 0,5$; el número de hormigas 10 y ciclos, 200, los que en su conjunto representan una de las mejores combinaciones para estos parámetros. Como vemos aquí, el valor escogido para el parámetro β indica un mismo nivel de importancia para la información global y local. Otros valores más altos para este parámetro también fueron considerados ($\beta = 5$ y $\beta = 10$); sin embargo, estos valores indujeron en muchos casos a una convergencia acelerada hacia óptimos locales de SH_D . La figura 6.5 muestra una situación típica representada por las curvas de convergencia para una instancia de MISP cuyo mejor valor objetivo es 62. Las curvas

¹⁴Más comentarios con respecto a esto al final del capítulo.

están rotuladas (α, β) , con $\alpha = 1$ y $\beta = 1, 5, 10$ y muestran la manera en que SH_D acelera la convergencia a medida que se aumentan los valores de β . En consecuencia, se escogió el valor $\beta = 1$ en el estudio experimental en todas las instancias de MISP consideradas para evitar situaciones de convergencia prematura.

El vector que representa el rastro de feromona, como en los anteriores experimentos para MKP y SCP, es inicializado en forma aleatoria de manera tal que $\tau_i \in (0, 1)$, $i \in \{1, \dots, n\}$.

Tres grupos de instancias fueron consideradas para este problema, el primer grupo fue generado acorde a dos métodos usados por Khuri et al. [12]. El primer método consiste en un algoritmo (ver figura 6.3.4) que genera grafos con n nodos, densidad $d \in \{0,1, 0,2, 0,3, 0,4, 0,5\}$ y un conjunto independiente de máxima cardinalidad m . Las instancias generadas de acuerdo a dicho algoritmo son denotadas por $Mn-d-m$.

El segundo método, dentro de este primer grupo de grafos, construye un grafo *escalable* (ver figura 6.7) el cual consiste de un número n (par) de nodos con $n > 6$. Si n es múltiplo de 4, la instancia contendrá dos máximos globales equivalentes de valor $|V^*| = n/2$. Estas soluciones particionan el conjunto de vértices en aquellos que tienen numeración

Figura 6.6: Algoritmo para la generación de grafos aleatorios el cual pre-selecciona un conjunto independiente de tamaño k

```

seleccionar aleatoriamente  $V^* = \{i_1, \dots, i_m\} \subseteq V = \{1, \dots, n\}$ 
for i=1 to n do
  for j=i+1 to n do
    if  $((Rnd(0, 1) < d) \text{ and } ((i \notin V^*) \text{ or } (j \notin V^*)))$ 
      then  $e_{ij} = 1$ 
      else  $e_{ij} = 0$ 
  done
done
```

par y aquellos que tienen numeración impar. En caso contrario, existe un único máximo global y está dado por $V^* = \{1, 3, \dots, n/2, n/2 + 1, \dots, n\}$, con $|V^*| = n/2 + 1$. Este tipo de grafos son denotados con *escal*- n , donde n es la cardinalidad del conjunto de vértices.

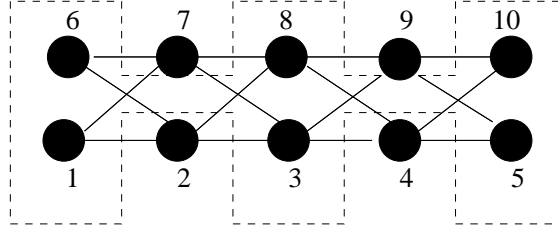


Figura 6.7: Un grafo con $n = 10$ nodos donde $V^* = \{1, 3, 5, 6, 8, 10\}$ (indicado con líneas punteadas).

El segundo grupo de instancias MISP fue generado siguiendo el proceso descrito en [25, 145, 144]. Este proceso selecciona cada $e_{ij} = 1$ (de manera similar al algoritmo 6.3.4) solamente si $\text{Rnd}(0, 1) \leq p$, donde p es la probabilidad de la existencia de un arco que conecte a los nodos i y j respectivamente. Este grupo de instancias corresponde a la familia de grafos aleatorios no dirigidos $G_{|V|,p}$ donde $n = |V|$. Más adelante se analizará en detalle el número esperado de conjuntos independientes que pueden ser encontrados en esta familia de grafos construidos según el proceso previamente descrito.

Finalmente, el tercer grupo de instancias fue obtenido del repositorio DIMACS [43]. De hecho, este grupo de instancias corresponde a instancias del problema de Clique máximo, pero dado que el Clique máximo se reduce al MISP¹⁵ es posible obtener instancias de MISP para las cuales el tamaño del conjunto independiente máximo es equivalente la grafo modificado según el proceso de reducción. Luego, los resultados obtenidos a través de la aplicación de algún método específico para MISP, pueden ser comparados con aquellos resultados obtenidos para las mismas ins-

¹⁵SI $G = \langle E, V \rangle$ es un grafo acíclico y $N \leq |V|$ es el clique de máxima cardinalidad para G , luego N representa la cardinalidad del conjunto independiente de máxima cardinalidad para el grafo $\bar{G} = \langle \bar{E}, V \rangle$.

tancias (sin reducir) pero usando una técnica para resolver el máximo Clique [3].

Los resultados para los tres grupos son mostrados en tablas organizadas como sigue: MVE representa la cardinalidad del conjunto independiente máximo encontrado: Prom(MVE) es el promedio respectivo de 30 corridas y #hits representa el número de resultados exitosos. Las columnas CiM y TiM muestran respectivamente el número promedio mínimo de ciclos para obtener el mejor resultado y el tiempo promedio transcurrido para obtener dicho resultado¹⁶.

La tabla 6.23 muestra los resultados obtenidos para las instancias de MISP correspondientes al primer grupo. Los resultados en esta tabla indican que el SH_D tuvo un muy buen comportamiento para todas las instancias del tipo $Mn-d-m$ para las cuales la solución óptima fue alcanzada en los primeros ciclos del algoritmo. Esto implica que hubo un mínimo o nada de cooperación entre las hormigas. El algoritmo sólo usó los valores iniciales del rastro (generados aleatoriamente) y la información de la heurística. La tabla 6.23 muestra adicionalmente, los resultados para este mismo grupo de instancias reportados por Khuri et al. [12]. Es en este trabajo donde se proponen los dos métodos de generación de instancias de MISP descritos anteriormente. El AE propuesto por Khuri et al. (llamado aquí AE_K) usa representación binaria y una técnica de penalización para manejo de restricciones en donde las soluciones (x) no factibles son penalizadas disminuyendo en n el valor objetivo por cada nodo no factible incluido en la solución:

$$f(x) = \sum_{i=1}^n (x_i - n \cdot x_i \cdot \sum_{j=i}^n x_j e_{ij}) \quad (6.7)$$

Muchos de los tamaños de instancias considerados aquí no fueron considerados en el artículo. Sin embargo, se incluyeron en la tabla 6.23 aquellos disponibles (últimas tres columnas)¹⁷. Las respectivas colum-

¹⁶Los tiempos están expresados en segundos.

¹⁷Para el grupo de instancias en común con el artículo de Khuri et al., se realizaron 100 corridas

nas MVE y Prom(MVE) para los algoritmos SH_D y AE_K , muestran una marcada diferencia en favor de SH_D , ya que todas las instancias en común fueron fácilmente resueltas por el algoritmo ACO. Por esta razón, se generaron instancias adicionales de MISP. Para las nuevas instancias, se obtuvieron igualmente los valores esperados. Notesé sin embargo, para los casos *escal** se observa una disminución en $\#hits$ y por ende en los valores promedios. Para estas instancias, el número de ciclos para alcanzar la mejor solución se incrementó de manera notable, es decir, se produjo una convergencia más lenta lo cual implica que se realizó un proceso de cooperación entre las hormigas.

Para el segundo grupo hemos considerado un conjunto de instancias de la familia de grafos $G_{|V|,p}$ [25] con diferentes valores de $n \in \{200, 400, 600\}$, el tamaño del problema y los valores de probabilidades $p \in \{0,2, 0,5, 0,6, 0,85, 0,9\}$, es decir que para cada tamaño se generaron instancias con diferentes grados de densidad. Para este tipo de grafos aleatorios es posible calcular el número esperado de conjuntos independientes de tamaño $1 \leq k \leq n$ [25]. En este sentido, se define la variable estocástica X_K que denota el número esperado de conjuntos independientes de tamaño k (para una instancia particular de un grafo aleatorio). Las tablas 6.24 y 6.25 muestran el número esperado de conjuntos independientes máximos de un determinado tamaño (X_k) que pueden ser encontrados en un grafo de tamaño $|V|$ y generado con el parámetro p .

Por ejemplo, para un grafo con $|V| = 200$ y $p = 0,2$, podemos determinar que el valor esperado para $E(X_{25}) = 382$ y para $E(X_{26}) = 9$. Esto significa que se espera que el grafo tenga alrededor de 382 y 9 conjuntos independientes de tamaño 25 y 26, respectivamente. Para el mismo ejemplo, el valor esperado para X_k con $k = 27$ es cercano a 0, y en el otro extremo, para $k \leq 24$, el valor esperado se incrementa en forma muy significativa. En otras palabras, conjuntos independientes de

por cada una de ellas para replicar el estudio de los autores.

tamaño 27 son muy raros e inclusive pueden no existir, mientras que conjuntos independientes de tamaño 24 son abundantes. Una situación similar puede ser observada para otras combinaciones de $|V|$ y p . Por lo tanto, esperamos que cualquier algoritmo que tenga un buen desempeño sobre esta clase de instancias debería obtener resultados tan cercanos como sea posible a los valores más grandes de k para cada combinación de $|V|$ y p mostrados en las tablas 6.24 y 6.25.

Cada instancia de este grupo es denotada como $Mn-p$ donde n es el número de nodos y p es la probabilidad usada en el proceso de generación de la misma. El número total de instancias consideradas en el estudio fue de 75 dada por la combinación de valores $n \in \{200, 400, 600\}$ y $p \in \{0,2, 0,5, 0,6, 0,83, 0,9\}$ ¹⁸ y para cada combinación de n y p fueron generados 5 grafos aleatorios usando diferentes semillas para el generador de números aleatorios. La tabla 6.26 muestra los resultados para dicho grafos generados aleatoriamente. La segunda columna, $N_{(k,r)}$, muestra para cada instancia cuatro pares (k, r) donde cada par indica que el SH_D encontró r -veces de 50 corridas de AE_D ¹⁹, un conjunto independiente de tamaño k . El valor de k correspondiente al par ordenado de más a la izquierda, representa el tamaño del conjunto independiente con menos probabilidad de existir en el grafo. En las columnas subsecuentes, k es decrementado en 1 hasta el valor $k - 3$, es decir el tamaño de conjuntos independientes que abundan en dicho grafo.

Para este grupo, el mejor desempeño de SH_D es logrado para todos los grafos para los cuales $p \in \{0,5, 0,6, 0,83, 0,9\}$. Es importante hacer notar que a pesar del tamaño del grafo y considerando los valores anteriores de p , se verifica que SH_D , o bien encontró soluciones con un tamaño igual al de los conjuntos independientes más raros, o muy cercanos a éste. Por otro lado, los grafos generados usando $p = 0,2$ fueron los que presentaron mayor dificultad, similarmente a los otros valores.

¹⁸Los mismos valores fueron usados en [145, 144].

¹⁹Se realizó para este grupo de instancias un experimento similar al publicado por Resende et al. [145, 144] debido a las características ya mencionadas de estos grafos

Esto se verificó para todos los grafos considerados independientemente del tamaño de cada uno de ellos. Por ejemplo, para las instancias de tamaño 200, 400 y 600, el desempeño es aun aceptable. Sin embargo, puede observarse (ver tabla 6.26) que los conjuntos independientes encontrados en general estaban dentro de los más abundantes. De esta manera, para las instancias $M200-0.2$, SH_D encontró una solución de tamaño 26, 29 de tamaño 25, 18 de tamaño 29 y así sucesivamente. Una situación similar se da para $M400-0.2$ y $M600-0.2$. Por otro lado, es notable la diferencia en relación al valor en las columnas CiM y TiM entre los grafos generados con $p = 0,2$ y aquellos generados con $p \in \{0,5, 0,6, 0,83, 0,9\}$. Tal como fue mencionado anteriormente, este tipo de instancias fueron consideradas por Resende et al. [145, 144]. La meta-heurística usada fue GRASP para el siguiente tipo de instancias: $M400-0.6$, $M400-0.83$, $M600-0.83$, $M1000-0.2$, $M1000-0.5$, $M1000-0.83$, $M2000-0.5$, y $M3500-0.5$. Los resultados mostrados en el estudio anterior son comparables a los obtenidos por SH_D para los tamaños 400 y 600. Para grafos más grandes (los resultados no son mostrados aquí), el SH_D fue probado con grafos $M1000-0.2$, $M1000-0.5$, $M1000-0.83$, $M2000-0.5$, y $M3500-0.5$. Nuevamente, se verifica que los casos de prueba más difíciles están representados por grafos de densidad baja ($M1000-0.2$) para los cuales SH_D nunca encontró los valores 38 o 37, mientras que GRASP para el mismo tipo de grafos sí lo hizo. Los valores encontrados para esta instancia fueron $\{(36, 5), (35, 14), (34, 24), (33, 7)\}$ ²⁰ Para las instancias restantes del último grupo, SH_D se comportó de una manera similar como con los otros grafos de media y alta densidad.

Los resultados correspondientes al tercer grupo son mostrados en la tabla 6.27 en la cual se han incorporado dos columnas adicionales, OCH y CBH. La columna OCH (Optimized Crossover Heuristic) corresponde a los mejores resultados reportados por Aggarawal et al. [3] en el cual se aplicó un AG a un conjunto de instancias que incluyen aquellas mostra-

²⁰ Esta notación es idéntica a la usada en la tabla 6.26.

das en el tabla 6.27. La otra columna, CBH (Continuos Base Heuristic) corresponde a los resultados obtenidos para dichas instancias con esta heurística determinística (Gibbons et al. [70]). Esta heurística también fue referenciada en el trabajo de Aggarwal et al. [3] con la finalidad de comparar sus resultados.

En relación al comportamiento de SH_D , vamos a considerar la división de las instancias probadas en dos subgrupos. El primero incluye las instancias para las cuales el algoritmo obtuvo los resultados esperados. Estas instancias incluyen las del tipo *c-fat*, *johnson*, y *p-hat*. Además, es importante remarcar que para esas instancias las heurísticas OCH y CBH también se comportaron de manera similar, excepto en algunas pocas de ellas. Por ejemplo, SH_D se comportó mejor que OCH y CBH en la instancia *p-hat-500-3*; que CBH en la instancia *p-hat-700-3*; finalmente, que OCH en *p-hat-500-2* y *p-hat-700-1*. En este subgrupo podemos incluir a la instancia *keller5* para la cual SH_D y OCH llegaron al óptimo.

El segundo subgrupo requiere ser considerado en forma particular. Hay algunas instancias que no fueron resueltas óptimamente por SH_D , sin embargo, sí lo fueron por OCH (*san200_0.7_1*, *san400_0.7_1*, *san400_0.7_2*) aunque SH_D superó a CBH. Por otro lado, existen algunas instancias para las cuales ninguna solución óptima fue encontrada por ninguno de los métodos, es decir, ni por SH_D , OCH o CBH. De cualquier manera, para esas instancias (*brock400_2*, *brock400_4*, y *brock800**) SH_D tuvo un mejor desempeño que OCH. Además, para las instancias *sanr400_0.5*, *sanr400_0.7*, *brock400_1* y *brock400_3*, SH_D tuvo un mejor desempeño que OCH y CBH, respectivamente. Una excepción es *brock200_2* para la cual SH_D u CBH encontraron el mismo valor. Para la instancia MANN_a27, SH_D tuvo un comportamiento intermedio entre OCH y CBH, mientras que para MANN_a45, SH_D tuvo en menor rendimiento que OCH y CBH.

De manera similar a las instancias del primer grupo $Mn-p-m$, hemos observado una situación análoga en relación a la convergencia de SH_D .

Por ejemplo, ver las instancias $c-fat^*$, $johnson^*$, $san400^*$ y $MANN^*$. Sin embargo, la convergencia prematura no implica en estos casos la obtención de resultados de mala calidad para las instancias $c-fat^*$ y $johnson^*$ las cuales parecen fáciles de resolver más aun considerando los resultados para estas mismas instancias a partir de la aplicación de los algoritmos CBH y OCH, entre muchos otros.

Para finalizar con SH_D aplicado a MISP, concluimos que dicho algoritmo es efectivo y competitivo para este problema. Las instancias de MISP consideradas fueron generadas de acuerdo a métodos usados en trabajos previos y también un conjunto de instancias para el problema de Máximo Clique obtenidas del segundo desafío de DIMACS. Para algunos casos de los tres tipos de grupos considerados, sólo unos pocos ciclos fueron necesarios para obtener el valor óptimo o mejor conocido, según el caso. Con respecto a los grafos aleatorios usados aquí, es importante destacar que el mejor desempeño de SH_D fue para aquellos grafos generados con las más altas probabilidades — grafos altamente conectados — aunque no ocurrió lo mismo para los grafos de menor densidad en las conexiones. De cualquier manera, el desempeño de SH_D para esta clase de instancias fue tan bueno como el desempeño de GRASP según los resultados reportados para este método. Para las instancias de DIMACS, SH_D mostró un desempeño comparable a varios algoritmos específicos ya sea para MISP o para el Problema del Máximo Clique.

6.3.5. Análisis global acerca de SH_D

En la sección anterior hemos presentado algunos resultados de la aplicación de SH_D para tres problemas de subconjunto: MKP, SCP y MISP. Para cada uno de dichos problemas se consideró un importante conjunto de instancias obtenidas desde distintas fuentes (OR Library y DIMACS) o generadas siguiendo procesos especialmente diseñados los que permiten conocer *a priori* el valor objetivo de las soluciones óptimas o valores esperados para los óptimos.

La calidad de los resultados obtenidos varía de un problema a otro y también de una instancia a otra (tipo y/o tamaño) para un mismo problema. La comparación de SH_D con resultados de algunos algoritmos muestra no sólo la factibilidad de la metaheurística ACO para este tipo de problema, sino también que puede ser considerado competitivo en algunos casos. En este contexto es importante mencionar que la relevancia de los resultados alcanzados no sólo está dada por la buena calidad global, sino que además, SH_D es el primer algoritmo ACO propuesto para este tipo de problemas de optimización combinatoria.

Existen algunos puntos importantes a destacar respecto de ciertos parámetros de SH_D y el efecto de sus posibles valores sobre el desempeño del algoritmo (este punto será considerado en el próximo capítulo). Por el momento se puede destacar que el parámetro β (importancia de la información local) fue uno de los que requirieron mayor variación en sus valores a fin de mejorar el desempeño global de SH_D . Más precisamente, en algunos casos fue necesario incrementar los valores de β para lograr mejores resultados. Esto podría deberse a que el algoritmo, para ciertas instancias de algún problema en particular, necesite ser más explotativo o bien, la heurística definida podría no ser la adecuada o lo suficientemente potente. En este trabajo no se pretende evaluar la calidad de las heurísticas usadas. Por el contrario, se pretende definir heurísticas simples podría evitar un sesgo del algoritmo en el cual la exploración del espacio de búsqueda esté altamente guiado por la información local.

Para finalizar esta sección, se destaca por un lado la robustez de SH_D para manejar distintos problemas del mismo tipo (en nuestro caso de subconjunto) ya que sólo es necesario el procedimiento de cómputo de la información local y las condiciones que determinan si las restricciones son satisfechas. Por otro lado, SH_D es una versión muy simple de un algoritmo ACO, lo que implica que existe más de una posibilidad para realizar futuras extensiones del mismo a fin de hacerlo más competitivo con respecto a otros algoritmos y también la posibilidad de su aplicación

a un amplio espectro de problemas con características similares (en el próximo capítulo haremos referencia a este punto nuevamente).

Tabla 6.22: Desempeño de SH_D sobre instancias más difíciles de SCP. Se incluyen la comparación con GENEsYs y los valores reportados por P.Chu. [32]. ND: No Disponible

Instancia	Óptimo o MVC	SH_D			GENEsYs		
		MVE	Prom(MVE)	#hits	MVE	Prom(MVE)	#hits
A.1	253	258	259	0	259	260	0
A.2	252	255	256	0	261	263	0
A.3	232	236	238	0	237	240	0
A.4	234	239	241	0	236	238	0
A.5	236	243	246	0	237	240	0
B.1	69	69	69	30	76	78	0
B.2	76	76	76	30	79	80	0
B.3	80	80	80	30	83	85	0
B.4	79	80	80	71	0	86	0
B.5	72	72	72	30	76	80	0
C.1	227	237	239	0	234	238	0
C.2	219	225	227	0	221	222	0
C.3	243	249	251	0	259	263	0
C.4	219	230	232	0	226	228	0
C.5	215	219	220	0	219	221	0
D.1	60	63	65	0	63	66	0
D.2	66	67	67	0	67	68	0
D.3	72	74	75	0	77	78	0
D.4	62	62	63	15	65	66	0
D.5	61	61	62	20	63	64	0
E.1	5	5	5	30	6	6	0
E.2	5	5	5	30	7	7	0
E.3	5	5	5	30	7	7	0
E.4	5	5	5	30	6	6	0
E.5	5	5	5	30	7	7	0
NRE.1	29	30	31	0	31	32	0
NRE.2	30	30	31	20	35	38	0
NRE.3	27	28	29	0	29	30	0
NRE.4	28	28	30	10	30	32	0
NRE.5	28	28	29	15	30	31	0
NRF.1	14	14	14	30	16	16	0
NRF.2	15	15	15	30	16	16	0
NRF.3	14	14	14	30	16	16	0
NRF.4	14	14	14	30	16	16	0
NRF.5	13	13	13	30	16	16	0
NRG.1	176	186	188	0	191	194	0
NRG.2	155	161	164	0	164	170	0
NRG.3	166	175	177	0	174	180	0
NRG.4	168	180	183	0	183	186	0
NRG.5	168	180	185	0	178	181	0
NRH.1	64	67	68	0	75	77	0
NRH.2	64	67	69	0	75	78	0
NRH.3	59	62	64	0	67	69	0
NRH.4	58	61	62	0	70	72	0
NRH.5	55	58	59	0	72	73	0
CYC.06	60	60	60	30	64	67	0
CYC.07	144	150	155	0	157	159	0
CYC.08	348	354	360	0	377	382	0
CYC.09	816	835	340	0	880	895	0
CLR.10	25	25	26	15	26	27	0
CLR.11	23	23	24	10	29	31	0
CLR.12	26	27	29	12	26	29	3
CLR.13	26	28	31	0	ND	ND	ND

Tabla 6.23: Resultados de SH_D aplicado a diferentes grafos (primer grupo de instancias de MISP)

Instancia	SH_D							AE_K		
	MVE	Prom(MVE)	#hits	CiM		TiM		MVE	Prom(MVE)	#hits
				Min	Prom	Min	Prom			
M100-0.1-45	46	46.00	100	5	9	0.14	0.30	47	37.39	1
M100-0.2-45	45	45.00	100	1	2	≈ 0	0.05	45	37.25	34
M100-0.3-45	45	45.00	100	1	1	≈ 0	≈ 0	45	41.38	77
M100-0.4-45	45	45.00	100	1	1	≈ 0	≈ 0	45	44.20	96
M100-0.5-45	45	45.00	100	1	1	≈ 0	≈ 0	45	44.72	99
M200-0.1-90	90	90.00	100	2	3	0.18	0.39	90	68.75	4
M200-0.2-90	90	90.00	100	1	2	≈ 0	0.20	90	81.05	54
M200-0.3-90	90	90.00	100	1	2	≈ 0	0.11	90	88.22	93
M200-0.4-90	90	90.00	100	1	2	≈ 0	0.03	90	90.00	100
M200-0.5-90	90	90.00	100	1	1	≈ 0	≈ 0	90	90.00	100
M300-0.1-135	135	135.00	30	1	2	0.47	0.84	ND	ND	ND
M300-0.2-135	135	135.00	30	1	2	≈ 0	0.51	ND	ND	ND
M300-0.3-135	135	135.00	30	1	2	≈ 0	0.57	ND	ND	ND
M300-0.4-135	135	135.00	30	1	2	≈ 0	0.20	ND	ND	ND
M300-0.5-135	135	135.00	30	1	1	≈ 0	≈ 0	ND	ND	ND
escal-100	50	50.00	30	4	24	0.09	0.70	ND	ND	ND
escal-200	100	99.20	21	6	70	0.65	8.99	ND	ND	ND
escal-300	150	146.40	6	18	112	17.10	34.96	ND	ND	ND
scal-102	52	51	15	1	21	≈ 0	0.76	50	44.94	0
scal-202	102	99.8	3	9	63	1.08	8.39	96	88.90	0
scal-302	152	148.8	3	29	90	8.54	27.11	ND	ND	ND

Tabla 6.24: Algunos valores para las variables X_k en grafos aleatorios

$ V $	Valor de probabilidad p								
	0.2			0.5			0.6		
200	X_{26}	X_{25}	X_{24}	X_{11}	X_{10}	X_9	X_9	X_8	X_7
	10	382	$> 10^4$	11	638	$> 10^4$	6	397	$> 10^4$
400	X_{31}	X_{30}	X_{29}	X_{13}	X_{12}	X_{11}	X_{11}	X_{10}	X_9
	15	995	$> 5 \cdot 10^4$	3	402	$> 2 \cdot 10^4$	≈ 0	32	3116
600	X_{34}	X_{33}	X_{32}	X_{14}	X_{13}	X_{12}	X_{11}	X_{10}	X_9
	16	1522	$> 10^5$	3	609	$> 5 \cdot 10^4$	11	1913	$> 10^5$

Tabla 6.25: Algunos valores para las variables X_k en grafos aleatorios

$ V $	Valor de probabilidad p					
	0.83			0.9		
200	X_6	X_5	X_4	X_5	X_4	X_3
	≈ 0	51	1561	≈ 0	65	1313
400	X_7	X_6	X_5	X_5	X_4	X_3
	≈ 0	16	1678	8	1051	$> 10^4$
600	X_7	X_6	X_5	X_6	X_5	X_4
	≈ 0.5	181	$> 10^4$	≈ 0	64	5346

Tabla 6.26: Resultados de SH_D aplicado a diferentes grafos aleatorios (segundo grupo de instancias de MISP)

Instance	SH_D							
	$N_{(k,r)}$				CiM		TiM	
					Min	Prom	Min	Prom
M200-0.2	(26,1)	(25,29)	(24,18)	(23,2)	0.88	11.48	5	51.8
M200-0.5	(11,39)	(10,11)	(9,0)	(8,0)	≈ 0	5.47	1.8	28.46
M200-0.6	(9,49)	(8,1)	(7,0)	(6,0)	≈ 0	4.68	1.2	24.58
M200-0.83	(5,50)	(4,0)	(3,0)	(2,0)	≈ 0	0.31	1	1.79
M200-0.9	(5,10)	(4,40)	(3,0)	(2,0)	≈ 0	.31	1	18.08
M400-0.2	(31,0)	(30,6)	(29,29)	(28,15)	33	89	9.67	25.03
M400-0.5	(13,1)	(12,39)	(11,10)	(10,0)	8	41	1.9	15.8
M400-0.6	(11,2)	(10,21)	(9,27)	(8,0)	2	32	0.5	7.5
M400-0.83	(7,0)	(6,47)	(5,3)	(4,0)	1	17	≈ 0	4.83
M400-0.9	(5,40)	(4,10)	(3,0)	(2,0)	1	25	≈ 0	7.82
M600-0.2 ²¹	(34,1)	(33,5)	(32,22)	(31,14)	44	96	22.34	63.02
M600-0.5	(14,0)	(13,27)	(12,23)	(11,0)	7	68	2.5	34.01
M600-0.6	(11,9)	(10,41)	(9,0)	(8,0)	5	29	1.89	17.21
M600-0.83	(7,1)	(6,49)	(5,0)	(4,0)	2	7	≈ 0	2.73
M600-0.9	(6,6)	(5,44)	(4,0)	(3,0)	2	5	≈ 0	1.35

Tabla 6.27: Resultados de SH_D (tercer grupo de instancias de MISP - DIMACS).

Instancia	Tamaño	Opt o MVC	SH _D							OCH	CBH
			MVE	Prom(MVE)	#hits	CiM		TiM			
						Min	Prom	Min	Prom		
c-fat-200-1	200	12	12	12	10	1	1	≈0	≈0	12	12
c-fat-200-2	200	24	24	24	10	1	1	≈0	0.72	24	24
c-fat-200-5	200	58	58	58	10	1	1	≈0	≈0	58	58
c-fat-500-1	500	14	14	14	10	1	1	≈0	≈0	14	14
c-fat-500-10	500	126	126	126	10	1	1	≈0	≈0	126	126
c-fat-500-2	500	26	26	26	10	1	1	≈0	≈0	26	26
c-fat-500-5	500	64	64	64	10	1	1	≈0	≈0	64	64
johnson16-2-4	120	8	8	8	10	1	1	≈0	≈0	8	8
johnson32-2-4	196	16	16	16	10	1	1	≈0	≈0	16	16
johnson8-2-4	28	4	4	4	10	1	1	≈0	≈0	4	4
johnson8-4-4	70	14	14	14	10	1	1	≈0	≈0	14	14
p-hat-300-1	300	8	8	8	10	3	7	0.85	2.97	8	8
p-hat-300-2	300	25	25	25	10	5	9	2.10	5.88	25	25
p-hat-300-3	300	36	36	36	10	15	43	9.82	27.72	36	36
p-hat-500-1	500	9	9	9	10	6	21	5.68	23.15	9	9
p-hat-500-2	500	36	36	36	10	8	19	13.12	34.27	35	36
p-hat-500-3	500	≥ 50	50	49.6	6	17	58	29.52	105.92	49	49
p-hat-700-1	700	11	11	10.3	5	4	72	6.47	155.15	9	11
p-hat-700-2	700	44	44	44	10	11	42	38.79	164.10	44	44
p-hat-700-3	700	≥ 62	62	60.7	4	14	62	49.66	235.85	62	60
keller4	171	11	11	11	10	2	7	0.14	0.81	11	10
keller5	776	27	26	23.79	0	8	95	17.92	240.65	25	21
san200_0.7_1	200	30	30	29.6	9	25	102	5.16	12.80	30	15
san200_0.7_2	200	18	15	15	1	11	65	2.09	12.5	15	12
san200_0.9_1	200	70	70	60.9	6	13	98	2.82	23.27	70	46
san200_0.9_2	200	60	60	50.59	5	2	76	3.8	20.48	60	36
san200_0.9_3	200	44	44	38.29	2	13	64	2.94	15.71	36	30
san400_0.5_1	400	13	13	8.7	1	1	25	≈0	21.33	13	8
san400_0.7_1	400	40	40	28.6	2	35	116	12.9	115.01	40	20
san400_0.7_2	400	30	18	17.19	0	1	3	≈0	2.0	30	15
san400_0.7_3	400	22	16	15.5	0	1	1	≈0	21.85	16	14
san400_0.9_1	400	100	100	63	2	1	23	≈0	26.81	100	50
sanr200_0.7	200	18	18	17.8	8	4	83	0.66	17.33	18	18
sanr200_0.9	200	≥ 42	42	41.2	2	11	61	2.91	16.14	42	41
sanr400_0.5	400	13	13	12.4	4	7	75	5.18	55.54	12	12
sanr400_0.7	400	≥ 21	21	20.2	3	17	91	11.65	63.28	20	20
brock200_1	200	21	21	20.1	4	5	27	1.08	6	21	20
brock200_2	200	12	12	10.7	6	1	4	≈0	3.72	11	12
brock200_3	200	15	15	13.9	5	5	50	1.8	8.02	14	14
brock200_4	200	17	17	15.3	1	6	63	0.82	10.32	16	16
brock400_1	400	27	25	23.9	0	7	56	4.55	36.2	24	23
brock400_2	400	29	25	23.9	0	17	101	13.01	79.13	24	24
brock400_3	400	31	25	23.9	0	37	87	28.08	67.29	24	23
brock400_4	400	33	26	24	6	28	56	20.29	40.46	24	24
brock800_1	800	23	20	19.2	20	15	59	16.24	68.36	19	20
brock800_2	800	24	20	19.5	0	12	61	12.87	70.47	19	19
brock800_3	800	25	20	18.86	0	21	68	23.5	78.72	19	20
brock800_4	800	26	20	18.9	0	18	56	19.88	64.08	19	19
MANN_a27	378	126	126	124.1	0	12	100	10.51	94.59	126	121
MANN_a45	1035	345	339	337.6	0	120	1000	146.86	768.77	343	336

Capítulo 7

Algoritmos Evolutivos y Operadores de Frontera

7.1. Introducción

En este capítulo presentamos el estudio experimental correspondiente a la aplicación del enfoque evolutivo al conjunto completo de problemas descritos en la sección 5.7. Los algoritmos evolutivos para problemas continuos y discretos serán denominados de aquí en adelante, AE_C y AE_D respectivamente.

Los algoritmos AE_C y AE_D manipulan ambos una población de soluciones completas. En este sentido, AE_C es similar a SH_C , dado que en ambos algoritmos las soluciones son representadas por un par de vectores en punto flotante. En el caso de AE_D , las soluciones son representadas por una cadena binaria. Aquí hay una diferencia substancial con SH_D dado que este último construye las soluciones paso a paso, mientras que AE_D , genera una población inicial de soluciones factibles y luego aplica los operadores genéticos especializados en el proceso de evolución. La principal característica de estos operadores es que sólo generan soluciones factibles sobre la frontera y para ello, incluyen información del problema para construir acordemente cada solución generada. En consecuencia, similarmente a lo explicado en el capítulo anterior para SH_C y SH_D , sólo es necesario realizar pequeños ajustes para poder aplicar

AE_C y AE_D para los respectivos problemas continuos y discretos.

En cuanto al algoritmo AE_C , los cambios a realizar son exactamente los mismos que los realizados para cada problema continuo usando SH_C . Para los problemas discretos, se debe modificar en AE_D lo siguiente: el proceso de inicialización de la población, las funciones `seleccionar()` y `actualizar()` invocadas en el operador de crossover (ver figura 5.10) y por último, la función `seleccionar_para_eliminar`, invocada en el operador de mutación (ver figura 5.11).

El enfoque de frontera implementado a través de AE_C también requiere de alguna técnica complementaria para manejo de restricciones tal como fue necesario para SH_C . De hecho, para AE_C se usa exactamente el mismo criterio seguido anteriormente, esto es, incorporar una técnica sencilla de penalización de manera tal que se pueda visualizar más claramente el comportamiento del algoritmo propuesto.

El objetivo principal de este capítulo es mostrar la factibilidad de la aplicación del enfoque de frontera en el contexto de los algoritmos evolutivos. En esta dirección se realizan comparaciones indirectas con otros algoritmos aplicados al mismo conjunto de problemas a los efectos de determinar la calidad y/o competitividad de AE_C y AE_D .

Cabe señalar que los problemas continuos y discretos estudiados en este capítulo son los mismos que los usados en los experimentos con los algoritmos ACO (SH_C y SH_D). Es decir, los problemas $G01$ a $G17$ y las instancias descritas en el capítulo anterior para MKP, SCP y MISP, respectivamente.

La siguiente sección presenta los resultados para los problemas continuos junto con un breve análisis de los resultados obtenidos por AE_C para cada uno de ellos. Esta sección culmina con un análisis global de AE_C en relación a su desempeño sobre los problemas estudiados. A continuación (sección 7.3) se presentan en primer término las modificaciones necesarias para adecuar AE_D a cada uno de los problemas discretos para luego realizar el análisis de los resultados sobre el conjunto de instancias

escogidas para el estudio. Esta sección finaliza también con un análisis global del desempeño de AE_D sobre los problemas y respectivas instancias estudiadas.

7.2. Problemas Continuos (aplicación de AE_C)

Las tablas mostradas en esta sección están organizadas de la misma manera que en la sección 6.2. Para una descripción detallada de dicha organización, remitirse a dicha sección.

Los valores de los parámetros de AE_C se describen a continuación: tamaño de la población 50 (un número estándar para el cual mostró un desempeño global satisfactorio); selección por torneo con $q = 2$ y una política de reemplazo global de la población; $p_c = 0,75$ (un valor estándar para la probabilidad de aplicación del operador de crossover); $p_m = 0,2$ (un valor relativamente alto); elitismo (la mejor solución hasta el momento es incorporada en cada nueva población generada), número de generaciones igual a 30000. El número de generaciones necesarias para obtener los mejores valores varía según la dificultad del problema y/o desempeño del algoritmo. Si bien el número máximo es alto, en cada tabla se muestra el número de evaluaciones promedio para encontrar los mejores valores. De esta manera se puede visualizar independientemente del número de generaciones, el esfuerzo realizado por el algoritmo. Los valores $\mu(t)$ son incorporados en cada tabla correspondiente a los problemas que necesiten usar un factor de penalización.

Problema G01

De las 9 restricciones de este problema, 6 están activas en el óptimo: 1,2,3,7,8, y 9. Si se observa la tabla 7.2, vemos que la solución óptima (con valor objetivo -15.00) fue encontrada cuando AE_C realizó la búsqueda sobre dichas restricciones. Sin embargo, los valores respectivos de la columnas Media y #Factibles evidencian que no en todas las ejecuciones

Tabla 7.1: Valor de penalización usado para G01 es $\mu(t) = 500$ para $t = 0, 1, \dots, t_{max}$.

Restricción	Función G01 (-15.00)					
	Mejor	Media	DesvEstand	Peor	# Factibles	Prom(#E)
1	-15.00	-14.95	0.09	-14.61	18	8309
2	-15.00	-14.84	0.12	-14.53	15	8420
3	-15.00	-14.92	0.28	-14.44	17	7030
4	-13.82	-13.53	≈ 0	-13.48	7	7320
5	-13.82	-13.56	0.56	-13.22	16	8688
6	-13.82	-12.82	3.76	-12.62	17	8490
7	-15.00	-13.53	1.15	-11.63	25	8132
8	-15.00	-13.77	1.11	-10.08	19	7750
9	-15.00	-13.82	1.02	-11.37	20	8406
Activas	-15.00	-15.00	0	-15.00	27	10540
Todas	-15.00	-15.00	0	-15.00	22	10390

alcanzó dicho valor ni tampoco todas las soluciones encontradas fueron factibles. Esta situación cambia cuando AE_C se ejecuta con las opciones de búsqueda “Activas” y “Todas”. Con ambas opciones, el algoritmo alcanza los mejores valores en todas las corridas, aunque no todas las soluciones encontradas son factibles, principalmente en la opción “Todas”. A diferencia de SH_D , se refleja aquí un incremento en el número de evaluaciones respecto de las opciones por búsqueda individual.

Problema G02

Los resultados para G02 son presentados para distintos valores de n , dado que es un problema escalable. La instancia más representativa de este problema es para $n = 20$ variables. En la tabla 7.2 se muestran los valores encontrados para G03 o función de Keane, con $n = 20, 50$ y 100 variables. Los mejores valores conocidos para cada una de las instancias son 0,803619 [83, 107], 0,831937 [157] y 0,845684. Este valor fue tomado de la tabla 6.2. El mejor resultado para $n = 20$ obtenido por AE_C no logró alcanzar el mejor valor conocido, aunque por una mínima diferencia de 0,000069. También se destaca un desempeño estable a lo largo de las 30 corridas, con muy pequeñas variaciones sobre los valores arrojados. Para $n = 50$, el resultado sobrepasó el mejor conocido [157] aunque no logró alcanzar el valor obtenido por SH_C de 0,835261. Una situación similar se verifica para $n = 100$. Como observación adicional, se destaca que para todas las instancias de este problema las soluciones

encontradas son factibles.

Tabla 7.2: Aquí se muestran los valores encontrados para la función de Keane con $n = 20, 50$ y 100 variables. Los mejores valores conocidos para cada uno de ellos son $0,803619$ [83, 107], $0,831937$ [157] y ND respectivamente, donde ND indica resultado no disponible.

Nro. de Variables	Función G02 (*)					
	Mejor	Media	DesvEstand	Peor	# Factibles	Prom(#E)
20	0.803550	0.803352	≈ 0	0.802849	30	32498
50	0.834863	0.833483	0.001	0.82988	30	40656
100	0.844686	0.841269	0.0017	0.838131	30	49300

Problema G03

Para este problema escalable con una restricción, AE_C alcanzó el valor óptimo para $n = 20$ y $n = 50$. Es claro que habiendo sólo una restricción ($h(x)$), el enfoque de frontera asegura que todas las soluciones obtenidas sean factibles. Es importante destacar que el enfoque de frontera propuesto aquí para problemas continuos no usa un operador genético especializado para mantener la factibilidad de las soluciones como en [120] para este problema y para G02 (función de Keane) en [157]. Por el contrario, el operador es relativamente generalizado, ya que recibe como parámetro la restricción sobre la cual realizar la búsqueda. Volveremos a retomar este punto más adelante de manera más detallada.

Tabla 7.3: G03 tiene una restricción $h(x)$ y no es necesario asignar valores a μ . Se muestran los resultados para $n = 20, 50$ variables

Nro. de Variables	Función G03 (1.0)					
	Mejor	Media	DesvEstand	Peor	# Factibles	Prom(#E)
20	1.0	1.0	0.0	1.0	30	16800
50	1.0	1.0	0.0	1.0	30	19200

Problema G04

El comportamiento de AE_C sobre G04 es similar a SH_D desde al menos dos perspectivas: a) sólo las opciones de búsqueda sobre las restricciones 1, 6 y Activas dieron soluciones factibles y, b) los resultados encontrados en cada una de las corridas coinciden con el valor óptimo

para dicho problema. Hay una pequeña diferencia en que SH_C mostró un comportamiento más robusto (ver desviación estándar).

Tabla 7.4: Valor de penalización usado para G04 es $\mu(t) = 8000000$ para $t = 0, 1, \dots, t_{max}$

Restricción	Función G04 (-30655.539)					
	Mejor	Media	DesvEstand	Peor	# Factibles	Prom(#E)
1	-30665.54	-30662.62	5.26	-30646.32	30	16660
2	-	-	-	-	-	-
3	-	-	-	-	-	-
4	-	-	-	-	-	-
5	-	-	-	-	-	-
6	-30655.54	-30665.52	0.01	-30665.47	30	12960
Activas	-30655.54	-30655.52	0.05	-30655.36	30	11154
Todas	-	-	-	-	-	-

Problema G05

Es importante destacar que para el problema G05, la búsqueda sobre la restricción 2 no arrojó soluciones factibles en ninguna de las 30 corridas. Si recordamos los resultados de SH_C para G05 (tabla 6.5) vemos que ocurría algo similar, con la diferencia que eran dos las restricciones que no arrojaron soluciones factibles, la 1 y 2. Si bien AE_C encuentra soluciones factibles sobre la restricción 2, éstas son pocas y de baja calidad en comparación con las otras opciones de búsqueda. En términos generales, se puede decir que AE_C obtiene muy buenos valores para G05. Aunque sin lograr alcanzar el valor óptimo, sólo los separa una pequeña diferencia. Con respecto al número de evaluaciones (#E), se puede observar que se mantienen similares para todo el conjunto de opciones de búsqueda. Sin embargo, el mayor número de soluciones factibles es obtenido con la opción “Activas”.

Tabla 7.5: Valor de penalización usado para G05 es $\mu(t) = 10$ para $t = 0, 1, \dots, t_{max}$.

Restricción	Función G05 (5126.49)					
	Mejor	Media	DesvEstand	Peor	# Factibles	Prom(#E)
1	5168.35	5170.22	0.99	5172.95	3	24212
2	-	-	-	-	-	-
3	5126.58	5130.53	5.8	5149.42	4	32860
4	5126.54	5130.29	9.31	5169.51	6	35080
5	5126.52	5128.41	4.08	5145.09	4	28200
Activas	5126.58	5129.80	7.26	5145.68	12	35400
Todas	5126.64	5130.12	3.84	5144.88	4	32320

Problema G06

Este problema no representa ninguna dificultad para AE_C . En cualquiera de las opciones de búsqueda dicho algoritmo alcanza el valor óptimo y adicionalmente, todas las soluciones de las 30 corridas corresponden a soluciones factibles. El número de evaluaciones es similar en todas las opciones.

Tabla 7.6: Valor de penalización usado para G06 es $\mu(t) = 10000$ para $t = 0, 1, \dots, t_{max}$

Restricción	Función G06 (6961.81)					
	Mejor	Media	DesvEstand	Peor	# Factibles	Prom(#E)
1	-6961.81	-6961.81	0.0	-6961.81	30	9004
2	-6961.81	-6961.81	0.0	-6961.81	30	9188
Activas	-6961.81	-6961.81	0.0	-6961.81	30	9000

Problema G07

Toda las opciones de búsqueda para este problemas dieron soluciones factibles y de muy buena calidad. Más precisamente, las opciones de búsqueda por las restricciones 2 y 4 son las que mejores valores arrojaron, aunque con pocas soluciones factibles. Dicha situación ocurre en la mayoría de las opciones de búsqueda por una restricción. En este sentido, la opción “Activas” no sólo arrojó los valores más cercanos al óptimo, sino que también encuentra una mayor cantidad de soluciones factibles. La búsqueda por las restricciones 7 y 8 (no activas en el punto óptimo) no dieron valores de buena calidad, como es predecible y de manera similar a lo ocurrido con el algoritmo SH_C para este problema.

Tabla 7.7: Valor de penalización usado para G07 es $\mu(t) = 10$ para $t = 0, 1, \dots, t_{max}$

Restricción	Función G07 (24.306)					
	Mejor	Media	DesvEstand	Peor	# Factibles	Prom(#E)
1	24.37	24.54	0.12	24.88	4	25400
2	24.37	24.55	0.23	25.05	5	37200
3	24.52	26.22	0.71	28.35	5	18080
4	24.38	25.40	2.69	39.45	3	36600
5	24.31	24.31	0	24.31	1	38080
6	24.32	25.29	2.43	35.32	2	32600
7	33.00	33.00	0	33.00	1	32200
8	41.06	41.06	0	41.06	1	30400
Activas	24.49	25.34	0.66	27.14	10	31080
Todas	24.46	25.45	0.46	26.78	6	24000

Problema G09

AE_C muestra un desempeño acorde con el enfoque de frontera para $G09$. Por un lado, logra resultados óptimos usando las opciones de búsqueda sobre las restricciones 1 y 4 (cercanos al óptimo), “Activas” y “Todas”. Esto es así dado que las restricciones 1 y 4 se encuentran activas en el óptimo. En el caso de la opción “Todas”, como se ha observado en todos los problemas estudiados, muestra un comportamiento similar a la opción “Activas”. Para las opciones 2 y 3, AE_C no encuentra soluciones factibles. Esta situación en SH_C no es la misma, aunque las soluciones encontradas por el algoritmo ACO no son de buena calidad.

Tabla 7.8: Valor de penalización usado para $G09$ es $\mu(t) = 10$ para $t = 0, 1, \dots, t_{max}$

Restricción	Función G09 (680.63)					
	Mejor	Media	DesvEstand	Peor	# Factibles	Prom(#E)
1	680.63	680.70	0.06	680.93	21	23000
2	-	-	-	-	-	-
3	-	-	-	-	-	-
4	680.75	682.06	1.48	684.27	19	22200
Activas	680.63	680.75	0.11	681.07	25	14800
Todas	680.63	681.67	0.65	683.78	21	30940

Problema $G10$

Nuevamente aquí, $G10$ se presenta como el caso más difícil para AE_C tal como lo fue para SH_C . En este sentido, ambos algoritmos mostraron un comportamiento similar, aunque la calidad de los resultados de AE_C están levemente por debajo de SH_C . También se puede observar para AE_C que sólo encuentra soluciones factibles cuando realiza la búsqueda usando alguna de las opciones que incluyen las restricciones activas en el óptimo. Particularmente la opción “Activas” es la que mejor calidad de las soluciones obtiene, dado por el Mejor, Media, Desviación estándar y número de evaluaciones.

Problema $G11$

El problema $G11$ no representa ningún grado de dificultad para AE_C , similarmente a lo ocurrido para SH_C y la mayoría de los métodos que lo incluyen como caso de prueba. Además, se observa un número muy bajo de evaluaciones. De hecho, mucho menor que la cantidad de evaluaciones

Tabla 7.9: Valor de penalización usado para G10 es $\mu(t) = 1,05 * \mu(t-1)$ y $\mu(0) = 200000$ para $t = 0, 1, \dots, t_{max}$

Función G10 (7049.331)						
Restricción	Mejor	Media	DesvEstand	Peor	# Factibles	Prom(#E)
1	7114.01	7526.39	404.03	8038.90	17	33200
2	7127.43	7584.32	424.33	8303.32	23	30838
3	7107.32	7553.40	785.04	10819.02	25	28074
4	-	-	-	-	-	-
5	-	-	-	-	-	-
6	-	-	-	-	-	-
Activas	7076.22	7179.03	59.23	7221.54	25	52440
Todas	7096.71	7397.16	191.66	7750.58	20	51120

realizadas por SH_C para este problema.

Tabla 7.10: G11 tiene una restricción $h(x)$ y no es necesario asignar valores a μ

Función G11 (0.75)						
Restricción	Mejor	Media	DesvEstand	Peor	# Factibles	Prom(#E)
Unica	0.75	0.75	0	0.75	30	566

Problema G13

El comportamiento global de AE_C para G13 es muy satisfactorio. La mejor calidad de los resultados, como se ha observado en los problemas previos, se verifica con la opción de búsqueda “Activas”. Por un lado, el mejor valor alcanzado es muy cercano al óptimo (diferencia mínima) y los resultados son muy similares entre sí. De entre las opciones de búsqueda por una restricción, la de menor nivel de desempeño global es restricción 1. Esto en términos de la diferencia entre el mejor y peor valor.

Tabla 7.11: Valor de penalización usado para G13 es $\mu(t) = 0,1$ para $t = 0, 1, \dots, t_{max}$

Función G13 (0.053950)						
Restricción	Mejor	Media	DesvEstand	Peor	# Factibles	Prom(#E)
1	0.054086	0.138544	0.11	0.338013	17	34344
2	0.054300	0.069974	0.048	0.239950	9	25800
3	0.054030	0.056380	0.015	0.112249	6	33600
Activas	0.053990	0.054480	0.0004	0.055380	15	32000

Por otro lado, este problema sigue presentando un grado relativamente alto de dificultad desde la perspectiva del número de evaluaciones requeridas y por el hecho que no en todas las corridas alcanzó soluciones factibles.

Problema G14

El problema G14 tiene dos restricciones del tipo $g(x)$ las cuales están activas en el punto óptimo. AE_C resuelve sin dificultades este problema usando cualquiera de las 3 opciones de búsqueda: 1, 2 y “Activas”. Se observa además que el número de evaluaciones se mantiene similar en todos los casos.

Tabla 7.12: Valor de penalización usado para G14 es $\mu(t) = 1$ para $t = 0, 1, \dots, t_{max}$

Restricción	Función G14 (50.80)					
	Mejor	Media	DesvEstand	Peor	# Factibles	Prom(#E)
1	-5.508013	-5.508013	0.0	-5.508013	30	1084
2	-5.508013	-5.508013	≈ 0.0	-5.5080	30	1148
Activas	-5.508013	-5.508013	0.0	-5.508013	30	1078

Problemas G15, G16 y G17.

Estos tres problemas tienen una única restricción ($h(x)$). En el caso de G15 se refleja un desempeño óptimo desde la perspectiva de la calidad de los resultados, robustez, número de soluciones factibles y una bajo número de evaluaciones. Este es otro caso para el cual es posible definir un operador de frontera *ad hoc* como se realizó en estudios previos con G02 y G03 dada la característica de la restricción a la que esta sujeto el problema G15. Los problemas G16 y G17 también fueron resueltos óptimamente, aunque se refleja un elevado número de evaluaciones y no en todas las corridas se pudo encontrar el valor óptimo (99,999). La restricción de estos dos problemas es la misma que la restricción del problema G01, por lo cual es posible usar el operador *esfera* según se definió en [120]. Retomaremos este punto más adelante para discutirlo en detalle. Por último, en todas las corridas se encuentran soluciones factibles para todos estos problemas (esto era algo de esperar dadas la características del método propuesto aquí y de los problemas en sí).

7.2.1. Análisis global de AE_C

Un conjunto de 15 problemas ampliamente conocidos y estudiados fueron considerados para la aplicación de AE_C , un algoritmo evolutivo

Tabla 7.13: G15, G16 y G17 tienen una restricción $h(x)$ y no es necesario asignar valores a μ

Restricción	Mejor	Media	DesvEstand	Peor	# Factibles	Prom(#E)
Función G15 (16.73889)						
Única	-16.73889	-16.73889	0.0	-16.73889	30	940
Función G16 (99.999)						
Única	99.999	99.100	1.05	96.717	30	52400
Función G17 (99.999)						
Única	99.999	99.760	0.51	97.43	30	52200

basado en el enfoque de frontera para problemas continuos. Muchas de las observaciones realizadas para SH_C pueden ser trasladadas a AE_D . Por ejemplo, el enfoque de frontera propuesto pudo ser incorporado en un algoritmo evolutivo de una manera muy sencilla, inclusive mucho más que lo fue para el algoritmo ACO. La razón es la siguiente: el diseño global de los algoritmos evolutivos es lo suficientemente flexible para manejar problemas de optimización de diferente tipo, ya sean continuos o discretos; aunque el manejo de restricciones no es una característica inherente a los mismos. Por el contrario, esta se logra en general con la incorporación de mecanismos como el enfoque de frontera propuesto aquí.

El enfoque de frontera implementado a través de AE_C , permite alcanzar los resultados óptimos o mejores conocidos en la mayoría de los 15 problemas considerados. También es destacable su similitud con el algoritmo SH_C en cuanto a su capacidad para ubicar soluciones sobre la frontera de las restricciones cuando éstas están activas.

Mostramos en esta sección (tabla 7.14) una comparación del desempeño de AE_S en relación a los resultados obtenidos por los dos algoritmos evolutivos usados en la sección 6.2.1: Runnarson & Yao [151] (llamado RY) y uno más reciente de Hamida & Schoenauer [83] (llamado HS¹). Para aquellos problemas que no fueron considerados por RY y/o HS se indica con ND (No Disponible). Vemos en la tabla 7.14 un desempeño muy satisfactorio de AE_C a lo largo de los 15 problemas estudiados. Similarmente a SH_C y a muchos otros algoritmos, además de RY y HS,

¹Mejor conocido como ASCHEA.

los problemas G01, G03, G11 y G15 fueron resueltos sin mayores complicaciones. Por otro lado, para los problemas G02 y G13 los resultados obtenidos distan mínimamente del mejor conocido y óptimo, respectivamente. El caso G07 sigue siendo el más difícil como regla general. Aunque su desempeño para este problema es bueno, está por debajo de RY, HS y SH_D . De cualquier manera, es importante destacar la calidad global de las soluciones a lo largo de las 30 corridas según se puede apreciar por la similitud de los respectivos valores medios y peores en cada caso, particularmente para aquellos problemas más difíciles como G07, G02 y G13; para los cuales los algoritmos RY y HS muestran mayores diferencias entre los valores mejores, medios y peores.

Tabla 7.14: Comparación de AE_C con los resultados de RY y HS

Prob.	Opt o MVC	Mejor Valor			Valor Medio			Peor Valor		
		AE_C	RY	HS	AE_C	RY	KM	AE_C	RY	HS
G01	-15.000	-15.000	-15.000	-15.000	-15.000	-15.000	-14.84	-15.000	-15.000	ND
G02	0.803619	0.803550	0.803515	0.803614	0.803352	0.781975	0.788949	0.822988	0.726288	ND
G03	1.000	1.000	1.000	1	1.000	1.000	0.99997	1.000	1.000	ND
G04	-30665.539	-30665.54	-30665.539	-30665.5	-30665.52	-30665.539	-30665.5	-30666.47	-30665.539	ND
G05	5126.498	5126.58	5126.497	5126.5	5126.80	5128.881	5126.53	5145.68	5142.472	ND
G06	-6961.814	-6961.81	-6981.814	-6961.81	-6961.81	-6875.940	-6961.81	-6961.81	-6350.262	ND
G07	24.306	24.49	24.307	24.3323	25.34	24.374	24.6636	27.14	24.642	ND
G09	680.630	680.63	680.63	680.63	680.63	680.56	680.641	681.07	680.763	ND
G10	7049.331	7076.22	7054.316	7049.33	7179.03	7559.192	7212.35	7221.54	8835.655	ND
G11	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75	ND
G13	0.053950	0.053990	0.053957	ND	0.054480	0.057006	ND	0.055380	0.216915	ND
G14	-5.5080	-5.5080	ND	ND	-5.5080	ND	ND	-5.5080	ND	ND
G15	-16.73819	-16.73819	ND	ND	-16.73819	ND	ND	-16.73819	ND	ND
G16	99.999	99.999	ND	ND	99.999	ND	ND	99.999	ND	ND
G17	99.999	99.999	ND	ND	99.999	ND	ND	99.999	ND	ND

7.3. Problemas discretos (aplicación de AE_D)

En la sección 5.4.2 se presentó el enfoque de frontera general para los operadores de crossover y mutación. Dichos operadores son definidos de manera general asumiendo una representación binaria para problemas de subconjunto.

Como prólogo al estudio experimental, se describen a continuación las componentes del algoritmo evolutivo que implementan: el proceso de inicialización de la población y los operadores genéticos para los problemas MKP, SCP y MISP. Estas componentes incluyen las funciones seleccionar, modificar y selec-

invocadas desde los algoritmos mostrados en las figuras 5.10 y 5.11.

7.3.1. AE_D para MKP

Se introducen en primer término, algunas definiciones usadas en las funciones invocadas desde los respectivos operadores genéticos. Debería tenerse en cuenta que los nombres de las variables usadas aquí se corresponden con la formulación de MKP (ver sección 5.7.2).

1. $\mu_i(\mathbf{x}) = \sum_{j=1}^n x_j r_{ij}$: La cantidad acumulada del recurso i para una solución \mathbf{x} .
2. $\gamma_i(\mathbf{x}) = c_i - \mu_i(\mathbf{x})$: La cantidad remanente del recurso i (disponibilidad) según la solución \mathbf{x} .
3. $\delta_{ij}(\mathbf{x}) = r_{ij}/\gamma_i(\mathbf{x})$: La razón entre la cantidad del recurso i usado por el proyecto j y la disponibilidad del recurso i teniendo en cuenta la solución \mathbf{x} .
4. $\bar{\delta}_j(\mathbf{x}) = \sum_{i=1}^m \delta_{ij}(\mathbf{x})/m$: Valor promedio del valor del item anterior sobre todos los recursos (restricciones) del problema.

■ Inicialización

Cada solución en la población inicial es generada en forma aleatoria. Para generar una solución sobre la frontera, se parte de un vector binario $\mathbf{x} = (0, \dots, 0)$ que representa el conjunto vacío. Cada componente del problema es elegido en forma aleatoria (distribución uniforme) y se va incorporando a la solución, es decir, el vector binario en la posición correspondiente pasa del valor 0 a 1. Este proceso se detiene cuando no es posible agregar más componentes a la solución, es decir, cuando en el intento de incorporar una componente, se viola al menos una de las restricciones. En otras palabras, se superó la capacidad de alguna de las mochilas.

■ Crossover

La funciones **seleccionar** y **modificar** son diseñadas de manera tal que toman en cuenta información de valores heurísticos relacionados con MKP. La selección de nuevas componentes es realizada a través de un proceso estocástico.

```

seleccionar (conjunto  $S$ , vector binario  $\mathbf{x}$  )
{
    para todo  $j \in S$ 
         $v_j = p_j / \bar{\delta}_j(\mathbf{x})$  /*  $p_j$  es el beneficio de
                                a componente  $j$ . */
    return  $j$  con probabilidad  $P(j) = \frac{v_j}{\sum_{k=1}^{|S|} v_k}$ 
}

```

```

modificar (conjunto  $S$ , int  $j$ , vector binario  $\mathbf{x}$ )
{
     $S = S - \{j\}$ 
    para todo  $k \in S$ 
        si (  $\exists i \mid \gamma_i(\mathbf{x}) < 0$  con  $x_k = 1$  )  $S = S - k$ 
}

```

■ Mutación

```

seleccionar_para_eliminar (conjunto  $D$ , vector binario  $\mathbf{x}$ )
{
    para todo  $j \in D$ 
         $v_j = \bar{\delta}_j(\mathbf{x})/p_j$ 
    return  $j$  con probabilidad  $P(j) = \frac{v_j}{\sum_{k=1}^{|D|} v_k}$ 
}

```

Mostraremos a continuación la aplicación de los operadores de frontera para MKP, tomando como base el ejemplo dado en la sección 5.7.2. Supongamos que $\mathbf{x} = 1011$ e $\mathbf{y} = 1100$ son los padres seleccionados para la aplicación del crossover. Es importante notar que después de la primera etapa, se obtiene una solución parcial $\mathbf{z} = 1 \bullet \bullet \bullet$, por lo tanto $S = \{2, 3, 4\}$. En la segunda etapa, el operador de crossover invoca por primera vez a la función **seleccionar** con $S = \{2, 3, 4\}$ y $\mathbf{z} = 1 \bullet \bullet \bullet$. Luego, los valores μ , γ , δ , and $\bar{\delta}$ deben ser calculados para obtener la probabilidad de selección de una nueva componente del conjunto S (para llevar la solución hacia la frontera).

1. La cantidad de cada recurso usado hasta el momento según la solución parcial en construcción:

$$\mu_1(\mathbf{z}) = \sum_{j=1}^4 z_j r_{1j} = r_{11} = 4$$

$$\mu_2(\mathbf{z}) = \sum_{j=1}^4 z_j r_{2j} = r_{21} = 6$$

$$\mu_3(\mathbf{z}) = \sum_{j=1}^4 z_j r_{3j} = r_{31} = 4$$

2. La cantidad disponible de cada uno de los recursos:

$$\gamma_1(\mathbf{z}) = c_1 - \mu_1(\mathbf{z}) = 8 - 4 = 4$$

$$\gamma_2(\mathbf{z}) = c_2 - \mu_2(\mathbf{z}) = 12 - 6 = 6$$

$$\gamma_3(\mathbf{z}) = c_3 - \mu_3(\mathbf{z}) = 10 - 4 = 6$$

3. Las respectivas razones entre los costos de los proyectos y la disponibilidad de todos los recursos:

$$\delta_{12}(\mathbf{z}) = \frac{4}{4} = 1 \quad \delta_{13}(\mathbf{z}) = \frac{2}{4} = 0,5 \quad \delta_{14}(\mathbf{z}) = \frac{1}{4} = 0,25$$

$$\delta_{22}(\mathbf{z}) = \frac{6}{6} = 1 \quad \delta_{23}(\mathbf{z}) = \frac{3}{6} = 0,5 \quad \delta_{24}(\mathbf{z}) = \frac{3}{6} = 0,5$$

$$\delta_{32}(\mathbf{z}) = \frac{4}{6} = 0,66 \quad \delta_{33}(\mathbf{z}) = \frac{2}{6} = 0,33 \quad \delta_{34}(\mathbf{z}) = \frac{3}{6} = 0,5$$

4. El promedio de los valores anteriores sobre todos los recursos:

$$\bar{\delta}_2(\mathbf{z}) = 0,886 \quad \bar{\delta}_3(\mathbf{z}) = 0,443 \quad \bar{\delta}_4(\mathbf{z}) = 0,416$$

Por lo tanto, los valores para las variable v_j son:

$$v_2 = \frac{10}{0,886} = 11,287 \quad v_3 = \frac{2}{0,443} = 4,515 \quad v_4 = \frac{6}{0,416} = 14,423$$

y los valores respectivos de probabilidad:

$$P(j = 2) = 0,378 \quad P(j = 3) = 0,139 \quad P(j = 4) = 0,483$$

Supongamos que la función **select** retorna el valor $j = 4$ basado en los valores $P(1)$, $P(2)$ y $P(3)$ anteriores. Luego, la función **modificar** eliminará el elemento 4 de S y también el elemento 2 (dado que la columna 2 deja de ser candidata debido a la incorporación de la columna 4 en la solución en proceso de generación). La segunda iteración de esta etapa comienza con $S = \{3\}$. En este caso, **seleccionar** retorna 3 y la solución final después de la aplicación del operador de crossover es $\mathbf{z} = 1001$.

7.3.2. AE_D para SCP

■ Inicialización

Este proceso es similar al usado para MKP, esto es, cada solución en la población inicial es generada en forma aleatoria incorporando una componente a la vez en una solución (inicialmente sin elementos). Sin embargo, el proceso termina cuando la solución es factible. En otras palabras, para MKP el vector $\mathbf{x} = (0, \dots, 0)$ (la solución de partida) es factible, pero no está en la frontera. La incorporación de elementos a \mathbf{x} , implica “llevar” la solución hacia la frontera, pero moviéndose siempre en el espacio factible. El caso de SCP es diferente, ya que el comienzo desde $\mathbf{x} = (0, \dots, 0)$ implica partir de una solución no factible y llevarla hacia la frontera hasta que se torne factible, es decir, hasta que todas las filas hayan sido cubiertas. Este proceso de generación puede dar lugar a soluciones redundantes en el sentido que se podrían eliminar algunas columnas sin tornar infactible a dicha solución.

■ Crossover

En lo que sigue se describen las funciones `seleccionar` y `modificar`. Las mismas son diseñadas teniendo en cuenta información del problema. Para este problema es importante remarcar que después de la finalización de la segunda etapa del operador de crossover y antes de la sentencia “return \mathbf{x} ”, todas las posibles columnas redundantes deberían ser eliminadas.

```

seleccionar (conjunto  $S$ , vector binario  $\mathbf{x}$  )
//  $CR_j$ : conjunto de filas cubierto por la columna  $j$ 
//  $U_{\mathbf{x}}$ : conjunto de filas no cubiertas hasta el
//      momento según  $\mathbf{x}$ 
//  $c_j$  : costo de la columna  $j$ 
{
    para todo  $j \in S$ 
         $v_j = \frac{|CR_j \cap U_{\mathbf{x}}|}{c_j}$ 
    return  $j$  con probabilidad  $P(j) = \frac{v_j}{\sum_{k=1}^{|S|} v_k}$ 
}

```

```

modificar (conjunto  $S$ , int  $j$ , vector binario  $\mathbf{x}$ )
{
     $S = S - \{j\}$ 
    para todo  $k \in S$ 
        si (  $CR_k \cap U_{\mathbf{x}} = \emptyset$  )  $S = S - k$ 
    }

```

■ Mutación

```

seleccionar_para_eliminar (conjunto  $D$ , vector binario  $x$ )
//  $U_x/j$ : conjunto de filas no cubiertas asumiendo que  $j$  fue
// eliminada de la solución  $\mathbf{x}$ .  $U_x/j \neq \emptyset$  dado que  $\mathbf{x}$  es una
// solución no redundante sobre la frontera .
{
para todo  $j \in D$ 

$$v_j = \frac{c_j}{|CR_j \cap U_x/j|}$$

return  $j$  con probabilidad  $P(j) = \frac{v_j}{\sum_{k=1}^{|D|} v_k}$ 
}

```

A continuación se explica la aplicación de los operadores de crossover y mutación (de frontera) siguiendo el ejemplo dado en la sección 5.7.2 para el problema SCP.

Supongamos que $\mathbf{x} = 0001100011$ y $\mathbf{y} = 010100001$ han sido seleccionados para realizar la recombinación. Las soluciones \mathbf{x} e \mathbf{y} están sobre la frontera y tienen valores objetivo 17 y 22 respectivamente.

Después de completar la etapa uno del operador de crossover, se obtiene una solución parcial $\mathbf{z} = \bullet 0 \bullet \bullet 1000 \bullet 1$. La etapa 2 comienza (ver figura 5.10, línea 9) con $S = \{1, 3, 4, 9\}$, es decir, las componentes en las cuales \mathbf{x} e \mathbf{y} no coinciden. La primera invocación a la función **seleccionar** (figura 5.10, línea 12) recibirá los siguientes argumentos: $S = \{1, 3, 4, 9\}$ y $\mathbf{z} = \bullet 0 \bullet \bullet 1000 \bullet 1$. Por lo tanto, $U_x = \{3, 6\}$, $CR_1 = \{3, 5\}$, $CR_3 = \{5, 6\}$, $CR_4 = \{3\}$, y $CR_9 = \{1, 6\}$. Así, los

respectivos valores para cada v_j son obtenidos como:

$$v_1 = \frac{|\{3,5\} \cap \{3,6\}|}{c_1} = \frac{1}{5} = 0,2,$$

$$v_3 = \frac{|\{5,6\} \cap \{3,6\}|}{c_3} = \frac{1}{7} = 0,1429,$$

$$v_4 = \frac{|\{3\} \cap \{3,6\}|}{c_4} = \frac{1}{2} = 0,5,$$

$$v_9 = \frac{|\{1,6\} \cap \{3,6\}|}{c_9} = \frac{1}{5} = 0,2,$$

y los respectivos valores de probabilidad:

$$P(j = 1) = 0,2/1,043 = 0,192 \quad P(j = 3) = 0,1429/1,043 = 0,137$$

$$P(j = 4) = 0,5/1,043 = 0,48 \quad P(j = 9) = 0,2/1,043 = 0,192$$

A pesar de que la columna $j = 4$ tenga asociado el mayor valor de probabilidad, las otras columnas aun puede ser seleccionadas. A través de este proceso estocástico, se le da mayor preferencia a las columnas que muestran una mayor razón entre el número corriente de filas que pueden ser cubiertas ($|CR_j \cap U_x|$) y el costo asociado a dicha columna (c_j). Suponiendo que la columna $j = 4$ fue seleccionada, el conjunto se actualiza, quedando $S = \{1, 3, 4, 9\} - \{4\} = \{1, 3, 9\}$. En el próximo paso (figura 5.10, línea 13) el conjunto de columnas candidatas es modificado. Por lo tanto, la función `modifica` eliminará todos los elementos de S que se convierten en redundantes después de la inclusión de $j = 4$. Para nuestro ejemplo, ninguna columna adicional a $j = 4$ puede ser eliminada de S . Para la próxima iteración, el conjunto de columnas es $S = \{1, 3, 9\}$. Supongamos ahora que `seleccionar` retorna 3 después del segundo llamado. Luego, $S = \emptyset$ debido a que las columnas 1 y 9 se convierten en redundantes para la solución después de la incorporación de la columna 3. La segunda etapa del operadores de crossover termina retornando la solución $\mathbf{z} = 0011100001$ con un costo 14.

Con el objeto de visualizar el proceso realizado por el operador de mutación, vamos a suponer que la solución $\mathbf{x} = 00011100011$ es el argumento

de dicho operador. También supongamos que `condicion_para_eliminar` (figura 5.11) determina que una componente de la solución será eliminada. Luego, la primer etapa de la mutación comienza con $D = \{4, 5, 9, 10\}$. Este es el valor del parámetro real cuando `condicion_para_eliminar` es invocada desde `seleccionar_para_eliminar` (figura 5.11, línea 10). Esta función, según la descripción previa, retorna la columna a ser eliminada de acuerdo a los siguientes valores:

$$\begin{aligned} CR_4 &= \{3\} & c_4 &= 2 & U_x/4 &= \{3\} \\ CR_5 &= \{1, 4\} & c_5 &= 8 & U_x/5 &= \{4\} \\ CR_9 &= \{1, 6\} & c_9 &= 5 & U_x/9 &= \{6\} \\ CR_{10} &= \{2, 5\} & c_{10} &= 2 & U_x/10 &= \{2, 5\} \end{aligned}$$

Luego, siguiendo la especificación en `seleccionar_para_eliminar` para SCP, se obtiene:

$$v_4 = \frac{2}{|\{1\}|} = 2 \quad v_5 = \frac{8}{|\{4\}|} = 8$$

$$v_9 = \frac{5}{|\{6\}|} = 5 \quad v_{10} = \frac{2}{|\{2,5\}|} = 1$$

y los respectivos valores de las probabilidades son:

$$P(j = 4) = \frac{2}{16} = 0,125 \quad P(j = 5) = \frac{8}{16} = 0,50$$

$$P(j = 9) = \frac{5}{16} = 0,3125 \quad P(j = 10) = \frac{1}{16} = 0,0625$$

Es claro que la columna 5 es una candidata a ser eliminada de la solución x . Aunque éste no es un procedimiento *greedy*, asumamos que `seleccionar_para_eliminar` retorna 5. Luego, la segunda etapa (figura 5.11, línea 16) comienza con el $S = \{1, 2, 3, 4, 6, 7, 8\}$. Ahora, el proceso es idéntico a la segunda etapa del operador de crossover, esto es, la solución no factible es completada a fin de obtener una solución factible sobre la frontera.

7.3.3. AE_D para MISP

■ Inicialización

Cada solución en la población inicial es generada de la misma manera que se procede en MKP. Se parte de una solución *vacía* y se le completa paso a paso hasta llegar a la frontera. Una solución que está sobre la frontera para MISP significa que no existen más nodos independientes en el grafo para ser agregados sin violar las restricciones.

- **Crossover** De manera similar a los otros problemas, se describen aquí las funciones **seleccionar** y **modificar** las cuales usan información relevante del problema.

```

seleccionar (conjunto  $S$ , vector binario  $\mathbf{x}$  )
//  $F_{add}(\mathbf{x}, j)$ : número de componentes factibles remanentes
// si  $j$  es agregado a la solución  $\mathbf{x}$ ,
// es decir, si se cambia de  $x_j = 0$  a  $x_j = 1$ .
{
  para todo  $j \in S$ 
     $v_j = F_{add}(\mathbf{x}, j) + 1$  /* se incrementa en 1
                                ya que todos los nodos
                                pueden ser candidatos */
  return  $j$  con probabilidad  $P(j) = \frac{v_j}{\sum_{k=1}^{|S|} v_k}$ 
}

```

```

modificar (conjunto  $S$ , int  $j$ , vector binario  $\mathbf{x}$ )
//  $U_j$ : conjunto de nodos no factibles debido a la inclusión
// del nodo  $j$  en la solución
{
     $U_j = \{i \in S - \{j\} \mid (j, i) \in E \vee (i, j) \in E\}$ 
     $S = S - \{j\} - U_j$ 
     $\mathbf{x}_i = 0$  para  $i \in U_j$ 
}

```

■ Mutación

```

seleccionar_para_eliminar (conjunto  $S$ , vector binario  $\mathbf{x}$ )
//  $F_{del}(\mathbf{x}, j)$ : número de componentes factibles
// si  $j$  es eliminado de la solución  $\mathbf{x}$ , es decir,
// si se cambia de  $\mathbf{x}_j = 1$  a  $\mathbf{x}_j = 0$ .
{
     $v_j = F_{del}(\mathbf{x}, j)$ 
    return  $j$  con probabilidad  $P(j) = \frac{v_j}{\sum_{k=1}^{|S|} v_k}$ 
}

```

El siguiente ejemplo ilustra la aplicación de los operadores de crossover y mutación según las funciones definidas previamente para MISP. El ejemplo está basado en la instancia de MISP dada en 5.7.2 y se asume que los padres seleccionados para la recombinación son $\mathbf{x} = 0010101011$ e $\mathbf{y} = 0011000110$. Los valores objetivos para \mathbf{x} e \mathbf{y} son 4 y 5 respectivamente. Una vez completada la primer etapa del crossover, se llega a la solución parcial $\mathbf{z} = 001 \bullet \bullet 0 \bullet \bullet 0 \bullet$. La segunda etapa (figura 5.10) comienza con el conjunto $S = \{4, 5, 7, 8, 10\}$, dado que $x_j \neq y_j$, para $j = 4, 5, 7, 8, 10$.

La primer invocación a **seleccionar** (figura 5.10, línea 12) recibirá los siguientes argumentos: $S = \{4, 5, 7, 8, 10\}$ y $\mathbf{z} = 001 \bullet \bullet 0 \bullet \bullet 0 \bullet$. De esta manera, se obtiene:

$$\begin{aligned} F(\mathbf{x}, 4) &= |\{8\}| = 1 & F(\mathbf{x}, 5) &= |\{7, 8, 10\}| = 3 \\ F(\mathbf{x}, 7) &= |\{5, 10\}| = 2 & F(\mathbf{x}, 8) &= |\{5, 7, 8\}| = 3 \\ F(\mathbf{x}, 10) &= |\{4, 10\}| = 2 \end{aligned}$$

y los respectivos valores heurísticos:

$$\begin{aligned} v_4 &= 2, & v_5 &= 4, & v_7 &= 3, \\ v_8 &= 4 & v_{10} &= 3 \end{aligned}$$

De esta manera, se obtienen los valores de probabilidad de selección para cada componente o nodo candidato:

$$P(j = 4) = \frac{2}{16} = 0,1250 \quad P(5) = \frac{4}{16} = 0,3750$$

$$P(j = 7) = \frac{3}{16} = 0,1875 \quad P(8) = \frac{4}{16} = 0,3750$$

$$P(j = 10) = \frac{3}{16} = 0,1875$$

En este caso, los vértices 5 y 8 tienen mayores oportunidades de ser parte de la solución resultante del operador. Asumamos que el nodo 8 es escogido, es decir, $z_8 = 1$. Por otro lado, la función **modificar** asigna $z_5 = 0$ y $z_7 = 0$, ya que estos nodos dejan de ser candidatos. De esta manera, la segunda iteración de esta primer etapa comienza con $S = \{4, 10\}$.

Ahora, los respectivos valores $F(S, j)$ para $j \in S$ son obtenidos:

$$F(\mathbf{x}, 4) = |\{10\}| = 1, \quad \text{y} \quad F(\mathbf{x}, 10) = |\{4\}| = 1$$

con

$$v_4 = 2, \quad \text{y} \quad v_{10} = 2.$$

y los respectivos valores de probabilidad son:

$$P(j = 4) = 0,5, \quad \text{y} \quad P(j = 10) = 0,5.$$

Asumamos ahora que el nodo 10 es el seleccionado. Después de este paso, $S = \emptyset$ y el operador de crossover termina el proceso. La solución obtenida es $\mathbf{z} = 0010000111$ con un valor objetivo 4.

El siguiente ejemplo nos muestra la aplicación del operador de mutación para MISP. Asumamos que la solución $\mathbf{x} = 0011000110$ es mutada. La primer etapa de este operador recibirá el conjunto $C = \{3, 4, 8, 9\}$, es decir, los posibles candidatos a ser eliminados de la solución \mathbf{x} . Luego, los valores correspondientes a $F_{elim}(\mathbf{x}, j)$ para $j \in C$ son los siguientes:

$$\begin{aligned} F_{elim}(\mathbf{x}, 3) &= |\{3\}| = 1 & F_{elim}(\mathbf{x}, 4) &= |\{1, 2, 5, 7, 10\}| = 6 \\ F_{elim}(\mathbf{x}, 8) &= |\{9\}| = 1 & F_{elim}(\mathbf{x}, 9) &= |\{7, 8\}| = 2 \end{aligned}$$

con,

$$v_1 = 1, \quad v_4 = 6, \quad v_8 = 1, \quad \text{y} \quad v_9 = 2,$$

y valores de probabilidad:

$$\begin{aligned} P(j = 1) &= 1/10 = 0,1; & P(j = 4) &= 6/10 = 0,6; \\ P(j = 8) &= 2/10 = 0,2 & \text{y} \quad P(j = 9) &= 1/10 = 0,1. \end{aligned}$$

Supongamos que `seleccionar_para_eliminar` retorna $j = 4$ (asumimos aquí que solamente una componente es eliminada cuando se aplica el operador de mutación). Luego, la segunda etapa comienza con $S = \{1, 10\}$. La función `seleccionar` puede retornar 1 o 10 con la misma probabilidad ya que $F_{agrega}(\mathbf{x}, 1) = F_{agrega}(\mathbf{x}, 10) = 0$, $v_1 = v_{10} = 1$, y $P(j) = P(10) = 0,5$. Si retorna el valor $j = 10$, la nueva solución después de la mutación es $\mathbf{x} = 0010000111$.

Sin embargo, si en cambio se seleccionan 3, 8, o 9 para su eliminación, la segunda etapa comenzará con $S = \emptyset$ y la función `chequear` retornará trivialmente *falso*. En consecuencia, la solución original es reconstruida y retornada tal cual (ver línea 25 en la figura 5.11).

7.3.4. Experimentos y resultados

En los experimentos realizados con AE_D se ha considerado el mismo conjunto de instancias estudiados bajo el algoritmo SH_D del capítulo anterior. Similarmente, los resultados se muestran comenzando con MKP, luego SCP y finalmente, MISP. Para finalizar esta sección se incluye un análisis general del comportamiento de AE_D sobre los problemas estudiados.

Las tablas mostradas en esta sección están organizadas de la manera similar a las encontradas en la sección 6.2 para cada uno de los problemas.

Los valores de los parámetros de AE_D se describen a continuación: tamaño de la población 20-100; selección por torneo con $q = 2$ y una política de reemplazo global de la población; $p_c = 0,75$ (un valor estándar para la probabilidad de aplicación del operador de crossover); $p_m = 0,1$; elitismo (la mejor solución hasta el momento es incorporada en cada nueva población generada), número de generaciones igual a 30000. De la misma manera que se estableció para AE_C , el número de generaciones necesarias para obtener los mejores valores varía según la dificultad de la instancia y/o desempeño del algoritmo. Si bien el número máximo es alto, en cada tabla se muestra el número de evaluaciones o generaciones promedio para encontrar los mejores valores. De esta manera se puede visualizar independientemente del número de generaciones, el esfuerzo realizado por el algoritmo.

Multiple Knapsack Problem

Las tablas 7.15 y 7.16 muestran dos grupos de instancias para el problema MKP. Ambos grupos fueron extraídos del repositorio OR Library [20]. Para el primer grupo de instancias $for1^*$ y $for2^*$, AE_D logró un desempeño muy bueno, excepto para dos casos, las instancias $for1-6$ y $for1-7$. Si bien encontró el óptimo para cada una de ellas, #hits no alcanza el 100 % de las 30 corridas. Cabe recordar que en el estudio de

Tabla 7.15: Resultados para las instancias for^* de MKP obtenidas de la OR Library

Instancia	Opt	AE_D						
		MVE	Prom(MVE)	#hits	GeM		TiM	
					Min	Prom	Min	Prom
for1-1	3800	3800	3800	30	1	1	0	0
for1-2	8706.1	8706.1	8706.1	30	1	1	0	0
for1-3	4015	4015	4015	30	1	1	0	0
for1-4	6120	6120	6120	30	1	1	0	0
for1-5	12400	12400	12400	30	1	1	0	0
for1-6	10584	10584	10576	18	2	382	0.01	2.71
for1-7	16507	16507	16494	3	98	303	1.27	3.89
for2-1	7772	7772	7772	30	7	68	0.22	2.29
for2-2	8722	8722	8722	30	42	213	1.81	8.76
for2-3	141278	141278	141278	30	61	255	0.18	0.72
for2-4	130883	130883	130883	30	21	239	0.04	0.38
for2-5	95677	95677	95677	30	25	106	0.04	0.73
for2-6	119337	119337	119337	30	1	1	0	0.02
for2-7	98796	98796	98786	30	4	17	0.01	0.04
for2-8	130623	130623	130623	30	50	236	0.13	0.56
for2-9	1095445	1095445	1095445	30	76	344	4.62	22.11
for2-10	624319	624319	624319	30	20	125	1.00	6.20
for2-11	4554	4554	4554	30	5	43	0.01	0.16

P.Chu [32] este grupo de instancias fueron consideradas, pero se decidió crear un nuevo grupo de instancias de mayor dificultad dado que el grupo existente en OR Library hasta ese momento no presentaba ninguna dificultad al algoritmo genético propuesto en ese momento. Aquí ocurre una situación similar.

Las instancias del segundo grupo son todas de tamaño $n = 100$ variables. Las primeras 10 tienen 5 restricciones y las restantes 10 restricciones (5.100^* y 10.100^*). Este grupo de instancias conforma un subconjunto de las instancias creadas por P.Chu para realizar sus estudios.

Las instancias 5.100^* fueron resueltas en su totalidad, excepto la instancia $5.100-04$, aunque con un valor muy cercano al mejor valor conocido (MVC). El incremento en el número de restricciones (mochilas) influye negativamente en el desempeño de AE_D (instancias 10.100^*). Por un lado es menor la cantidad de instancias resueltas. Por otro lado, se observa un mayor error porcentual para aquellas en que no se alcanzó MVC (particularmente para $10.100-08$ y $10.100-09$). Adicionalmente se observa un marcado incremento en el número promedio de generaciones necesarias para alcanzar los mejores valores (GeM) y por ende en los tiempos respectivos (TiM).

Tabla 7.16: Resultados para las instancias $(5+10).100^*$ de MKP obtenidas de la OR Library y generadas por Paul Chu[32]

Instancia	MVC	AE_D						
		MVE	Prom(MVE)	# hits	GeM		TiM	
					Min	Prom	Min	Prom
5.100-01	24381	24381	24377	24	28	528	1.28	22.81
5.100-02	24274	24274	24274	310	33	336	1.62	16.53
5.100-03	23551	23551	23541	93	57	336	2.68	15.90
5.100-04	23534	23503	23471	0	4	465	0.18	20.49
5.100-05	23991	23991	23953	31	233	547	64.76	150.75
5.100-06	24613	24613	24610	24	114	427	4.79	17.71
5.100-07	25591	25591	25523	6	18	305	0.86	12.98
5.100-08	23410	23410	23406	27	18	213	0.85	10.05
5.100-09	24204	24204	24185	9	201	533	9.85	25.46
5.100-10	24411	24411	24402	24	11	303	0.49	14.10
10.100-01	23064	23050	22962	0	62	424	19.21	130.74
10.100-02	22801	22801	22725	6	45	315	12.82	90.29
10.100-03	22131	22131	22030	3	151	405	46.78	125.48
10.100-04	22772	22772	22630	12	80	479	24.78	148.41
10.100-05	22731	22731	22702	3	98	560	30.36	173.50
10.100-06	22777	22777	22697	27	67	410	20.75	127.03
10.100-07	21875	21875	21858	24	41	515	6.44	81.00
10.100-08	22635	22551	22494	0	20	442	3.12	69.87
10.100-09	22511	22370	22309	0	74	439	11.91	71.50
10.100-10	22702	22702	22688	24	7	753	1.09	70.50

Set Covering Problem

Recordemos que las instancias consideradas para este problema corresponden a las tablas 6.18 y 6.19 descritas en el capítulo anterior las cuales pueden ser diferenciadas por tamaño y/o densidad. Desde el punto de vista de la densidad, están las de 20 % (E^* y NRF^*), 12 % (CLR^*), 10 % (NRE^*), 5 % (B^* , D^* , NRH^*) y alrededor de 2 % o menos (A^* , C^* , NRG^* , CYC^*). Aquí también hemos considerado el software GENEYS según fuera descrito previamente (T.Bäck [7]). De hecho, hemos incluido en la tabla 7.17 los mismos resultados mostrados en la tabla 6.22. Antes de analizar los resultados en particular, veamos a través de la figura 7.1 una descripción compacta de los resultados (expresados en término del error porcentual) encontrados por AE_D y GENEYS para las instancias agrupadas según su densidad. Claramente puede observarse para ambos algoritmos el incremento en el error porcentual a medida que disminuye la densidad de las instancias. Por otro lado es interesante destacar que AE_D para las instancias con densidad 20 % obtuvo los mejores valores para cada una de ellas. Una situación similar ocurre para las instancias con densidad 12 % y 10 %.

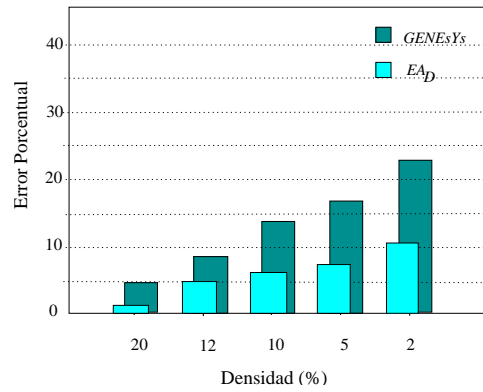


Figura 7.1: La figura muestra el promedio del error porcentual según Prom(MVE) para todas las instancias agrupadas por densidad.

Para las instancias con densidad 5% aun es capaz de alcanzar los óptimos (B^*) o valores muy cercanos a éstos (NRH^*). Los valores promedios para B^* y NRH^* evidencian un comportamiento robusto en las 30 corridas. Las instancias con densidad del 2% o menor, son las que representan las de mayor dificultad, con excepción de la instancia CYC.06 la cual es resuelta en dos de las 30 corridas.

La tabla 7.17 muestra en detalle los resultados para cada una de las instancias estudiadas con AE_D . Adicionalmente, se puede observar en esta tabla el incremento en el número de generaciones necesarias (GeM) para lograr los mejores resultados (MVEs) a medida que disminuye la densidad de las mismas.

Maximum Independent Set Problem

Los experimentos realizados con AE_D para MISP incluyen el conjunto de instancias (dividido en tres grupos) descrito en la sección 6.3.4 correspondiente a este problema. Brevemente volvemos a describirlos aquí:

- Primer grupo: instancias generadas acorde a dos métodos usados por Khuri et al. [12]. El primer método consiste en un algoritmo (ver figura 6.3.4) que genera grafos con n nodos, densidad $d \in \{0,1, 0,2, 0,3, 0,4, 0,5\}$ y un conjunto independiente de máxima cardinalidad m . Las instancias generadas de acuerdo a dicho algoritmo son

denotadas por $Mn-d-m$.

El segundo método construye un grafo *escalable* (ver figura 6.7) el cual consiste de un número n par de nodos con $n > 6$. Si n es múltiplo de 4, la instancia contendrá dos máximos globales equivalentes de valor $|V^*| = n/2$. Estas soluciones particionan el conjunto de nodos en aquellos que tienen numeración par y aquellos que tienen numeración impar. En caso contrario, existe un único máximo global y está dado por $V^* = \{1, 3, \dots, n/2, n/2 + 1, \dots, n\}$, con $|V^*| = n/2 + 1$. Este tipo de grafos son denotados con *escal- n* , donde n es la cardinalidad del conjunto de nodos.

- Segundo grupo: instancias generadas siguiendo el proceso descrito en [25, 145, 144]. Este proceso selecciona cada $e_{ij} = 1$ (de manera similar al algoritmo 6.3.4) solamente si $\text{Rnd}(0, 1) \leq p$, donde p es la probabilidad de la existencia de un arco que conecte a los nodos i y j respectivamente. Este grupo de instancias corresponde a la familia de grafos aleatorios no dirigidos $G_{|V|,p}$ donde $n = |V|$. El número esperado de conjuntos independientes de cierto tamaño para distintos valores de n y p fueron dados en la tabla 6.24.
- Tercer grupo: instancias obtenidas del repositorio DIMACS [43]. De hecho, este grupo de instancias corresponde a instancias del problema de Clique máximo, pero dado que el Clique máximo reduce a MIS² es posible obtener instancias de MIS para las cuales el tamaño del conjunto independiente máximo es equivalente al grafo modificado según el proceso de reducción.

La tabla 7.18 muestra los resultados del primer grupo, instancias propuestas por Khuri et al. [12] junto con los resultados publicados en su artículo. Vemos claramente que todas las instancias consideradas por

²Si $G = \langle E, V \rangle$ es un grafo acíclico y $N \leq |V|$ es el clique de máxima cardinalidad para G , luego N representa la cardinalidad del conjunto independiente de máxima cardinalidad para el grafo $\tilde{G} = \langle \bar{E}, V \rangle$.

Khuri et al. fueron resueltas sin ninguna dificultad por AE_D . Para las instancias $m100^*$, $m200^*$ y $m300^*$ (se realizaron 100 corridas como en el estudio de Khuri et al. [12]) sólo se observa una disminución de $\#hits$ con respecto a SH_D para las instancias con densidad $d = 0,1$. Algunas instancias del tipo *escal*^{*} también presentan cierto grado de dificultad para AE_D . Más precisamente, las instancias *escal300*, *escal2002* y *escal302*. Sin embargo los mejores valores arrojados están muy cerca del óptimo (ver también los valores promedios).

Los resultados para las instancias del segundo grupo (grafos aleatorios con valor esperado para el número de conjuntos independientes de cierto tamaño) son mostrados en la tabla 7.19. De la misma manera que lo realizado para SH_D , se realizaron 50 corridas de AE_D . Si se observa la columna 1 para todas las instancias, vemos que sólo 3 entradas aparecen con un 0 en la segunda componente. Es decir, en el resto de las instancias AE_D encontró al menos uno de los conjuntos independientes más escasos según la variable X_k (ver tablas 6.24 y 6.25). Vemos que la probabilidad p usada para generar los grafos y la cual determina su densidad, también influye en el desempeño de AE_D . En general, los más difíciles son aquellos generados con $p = 0,2$. Esto se refleja principalmente en el hecho de que tiende a encontrar conjuntos independientes más abundantes, particularmente para la instancia *m600-0.2*. Vemos para dicha instancia que de las 50 ejecuciones realizadas, hay algunas ejecuciones que no lograron ubicar alguno de los conjuntos independientes de mayor cardinalidad. Es decir, en estos casos, AE_D quedó atrapado en óptimos locales. Otra medida de la complejidad de este tipo instancias es el aumento del número de generaciones necesarias (GeM) para alcanzar los mejores valores en relación a las instancias generadas con un valor mayor de p .

Por último, se muestra en la tabla 7.20³ los resultados para el tercer grupo de instancias (DIMACS) y dos columnas de los mejores resultados obtenidos por los algoritmos OCH [3] y CBH [70], según se describieron

³Idéntica organización que la usada con SH_D .

en el capítulo anterior.

Las instancias $c-fat^*$ y $johnson^*$ fueron resueltas sin ningún inconveniente por AE_D en un número muy bajo de generaciones. Una situación similar se observa para las $Keller^*$, $san200^*$ y $san400^*$; aunque en promedio necesita un mayor número de generaciones para alcanzar los mejores resultados.

Las del grupo $p-hat^*$ fueron resueltas 3 de 9 instancias, $p-hat300-1$, $p-hat300-2$ y $p-hat500-1$. Para el resto se observan valores cercanos a los óptimos, pero menores a los obtenidos por OCH y CBH. Dos instancias $sanr^*$ fueron resueltas óptimamente y las otras dos, con valores muy cercanos al óptimo. MANN_a27 fue resuelta exitosamente, aunque para MANN_a45, se obtuvo un valor igual a OCH. Por último, esto es, en términos de la calidad de las soluciones, están las instancias $brock^*$, las que fueron más difíciles para AE_D . Sólo dos de ellas fueron resueltas, aunque los valores encontrados para las restantes son comparables (en muchos casos iguales) a los valores obtenidos por OCH y CBH.

7.3.5. Análisis global de AE_D

El estudio experimental mostrado en la sección anterior muestra un muy buen desempeño global de AE_D . Este algoritmo fue aplicado a distintas instancias de MKP, SCP y MISP. El enfoque de frontera implementado en AE_D permitió su aplicación a dichos problemas discretos restringidos sin necesidad de recurrir a complejos mecanismos para el manejo de restricciones que permitieron lograr una adecuada calidad en las soluciones.

De la misma manera que SH_D , el desempeño de AE_D no es idéntico para todos los problemas e instancias considerados. En el caso de MKP, AE_D fue capaz de alcanzar las soluciones óptimas en la mayoría de las instancias consideradas, sin embargo, su rendimiento decae cuando las instancias contienen un número incremental de restricciones (grupo 10.100^*).

Para SCP, su desempeño si vio influenciado principalmente por la densidad de los grafos dados como entrada, aunque el tamaño de la instancias no es un factor menor⁵. En este sentido cabe recordar que las instancias fueron pre-procesadas para obtener instancias equivalentes pero de menor tamaño. Finalmente, el desempeño de AE_D para las instancias consideradas de MISP refleja una calidad en los resultados comparables a otros algoritmos diseñados específicamente para este problema. Por cierto, el grupo de instancias consideradas de MISP es abundante y de diversa procedencia, muchas de ellas estudiadas a través de los métodos más variados (AEs, GRASP, Tabu Search, Versiones de búsqueda local, heurísticas *ad hoc*, etc.).

Por último, cabe destacar que AE_D se ejecutó con valores de parámetros estándares (en general) y qué sólo fue necesario cambiar un pequeño grupo de componentes para adecuarlo a diferentes problemas.

⁵Las instancias de SCP disponibles en la OR Library son en su mayoría de gran tamaño.

Tabla 7.17: Resultados del AE_D para un conjunto de instancias de SCP

Instancia	Opt o MVC	AE_D			GENEsYs		
		MVE	Prom(MVE)	#hits	MVE	Prom(MVE)	#hits
A.1	253	256	256.50	0	259	260	0
A.2	252	254	257.00	0	261	263	0
A.3	232	234	238.94	0	237	240	0
A.4	234	236	237.34	0	236	238	0
A.5	236	237	238.05	0	237	240	0
B.1	69	69	70.50	1	76	78	0
B.2	76	76	76.03	29	79	80	0
B.3	80	80	80.96	1	83	85	0
B.4	80	80	81.01	15	71	86	0
B.5	72	72	72.03	29	76	80	0
C.1	227	230	232.01	3	234	238	0
C.2	219	221	223.80	0	221	222	0
C.3	243	246	248.00	0	259	263	0
C.4	219	222	226.35	0	226	228	0
C.5	215	217	218.05	0	219	221	0
D.1	60	61	61.95	0	63	66	0
D.2	66	67	67.50	0	67	68	0
D.3	72	73	75.10	0	77	78	0
D.4	62	62	62.60	12	65	66	0
D.5	61	61	63.20	0	63	64	0
E.1	5	5	5	30	6	6	0
E.2	5	5	5	30	7	7	0
E.3	5	5	5	30	7	7	0
E.4	5	5	5	30	6	6	0
E.5	5	5	5	30	7	7	0
NRE.1	29	29	29.00	30	31	32	0
NRE.2	30	30	31.30	3	35	38	0
NRE.3	27	27	28.20	5	29	30	0
NRE.4	28	28	29.50	2	30	32	0
NRE.5	28	28	28.40	13	30	31	0
NRF.1	14	14	14.40	21	16	16	0
NRF.2	15	15	15.30	23	16	16	0
NRF.3	14	14	15.03	2	16	16	0
NRF.4	14	14	14.60	14	16	16	0
NRF.5	13	13	14.35	2	16	16	0
NRG.1	176	180	182.30	0	191	194	0
NRG.2	155	160	161.50	0	164	170	0
NRG.3	167	171	173.03	0	174	180	0
NRG.4	172	177	179.20	0	183	186	0
NRG.5	168	171	175.00	0	178	181	0
NRH.1	64	65	66.30	0	75	77	0
NRH.2	64	66	66.50	0	75	78	0
NRH.3	59	61	62.10	0	67	69	0
NRH.4	58	61	61.58	0	70	72	0
NRH.5	55	58	59.90	0	72	73	0
CYC.06	60	60	61.85	2	64	67	0
CYC.07	144	150	152.70	0	157	159	0
CYC.08	348	360	378.20	0	377	382	0
CYC.09	846	854	871.10	0	880	895	0
CLR.10	25	25	26.06	2	26	27	0
CLR.11	23	23	24.80	2	29	31	0
CLR.12	26	26	27.04	2	26	29	3
CLR.13	26	ND	ND	ND	ND	ND	ND

Tabla 7.18: Resultados para el conjunto de instancias de MISP generadas según los métodos propuestos por Khuri et al. [12]

Instancia	Opt	AE_D							AE_K		
			Prom(MVE)	#hits	GeM		TiM		MVE	Prom(MVE)	#hits
					Min	Prom	Min	Prom			
M100-0.1-45	45	45	44.5	65	1	669	0	0.95	47	37.39	1
M100-0.2-45	45	45	45	100	1	141	0	0.22	45	37.25	34
M100-0.3-45	45	45	45	100	1	1	0	0	45	41.38	77
M100-0.4-45	45	45	45	100	1	1	0	0	45	44.20	96
M100-0.5-45	45	45	45	100	1	1	0	0	45	42.72	99
M200-0.1-90	90	90	86.6	40	1	607	0	0.75	90	68.75	4
M200-0.2-90	90	90	90	100	1	18	0	0.03	90	81.05	54
M200-0.3-90	90	90	90	100	1	12	0	0.02	90	88.22	93
M200-0.4-90	90	90	90	100	1	1	0	0	90	90.00	100
M200-0.5-90	90	90	90	100	1	1	0	0	90	90.00	100
M300-0.1-135	135	135	130.26	18	1	240	0	0.96	ND	ND	ND
M300-0.2-135	135	135	135	30	1	80	0	0.19	ND	ND	ND
M300-0.3-135	135	135	135	30	1	1	0	0	ND	ND	ND
M300-0.4-135	135	135	135	30	1	1	0	0	ND	ND	ND
M300-0.5-135	135	135	135	30	1	1	0	0	ND	ND	ND
escal-100	50	50	49.4	23	1	170	0	0.16	ND	ND	ND
escal-200	100	100	98	1	1	1187	1.06	3.91	ND	ND	ND
escal-300	150	146	143	0	1	2100	0	3.81	ND	ND	ND
escal-102	52	52	50.33	5	1	192	0	0.14	50	44.94	0
escal-202	102	100	97	0	1	748	0	2.58	96	88.90	0
escal-302	152	146	142	0	1	2579	0	5.32	ND	ND	ND

Tabla 7.19: Resultados para el conjunto de instancias MISP generadas según el método descrito por Bollobas et al. [25]

Instance	AE_D							
	$N_{(k,r)}$				GeM		TiM	
					Min	Prom	Min	Prom
M200-0.2	(26,0)	(25,7)	(24,14)	(23,29)	2	11.48	5	51.8
M200-0.5	(11,7)	(10,40)	(9,3)	(8,0)	3	5.47	1.8	28.46
M200-0.6	(9,13)	(8,37)	(7,0)	(6,0)	1	4.68	1.2	24.58
M200-0.83	(5,50)	(4,0)	(3,0)	(2,0)	1	0.31	1	1.79
M200-0.9	(5,8)	(4,42)	(3,0)	(2,0)	≈ 0	.31	1	18.08
M400-0.2	(31,0)	(30,8)	(29,25)	(28,17)	33	89	9.67	25.03
M400-0.5	(13,1)	(12,30)	(11,19)	(10,0)	8	41	1.9	15.8
M400-0.6	(11,3)	(10,22)	(9,25)	(8,0)	2	32	0.5	7.5
M400-0.83	(7,0)	(6,45)	(5,5)	(4,0)	1	17	≈ 0	4.83
M400-0.9	(5,35)	(4,15)	(3,0)	(2,0)	1	25	≈ 0	7.82
M600-0.2 ⁴	(34,1)	(33,3)	(32,20)	(31,18)	44	96	22.34	63.02
M600-0.5	(14,0)	(13,30)	(12,20)	(11,0)	7	68	2.5	34.01
M600-0.6	(11,15)	(10,35)	(9,0)	(8,0)	5	29	1.89	17.21
M600-0.83	(7,1)	(6,40)	(5,9)	(4,0)	2	7	≈ 0	2.73
M600-0.9	(6,8)	(5,42)	(4,0)	(3,0)	2	5	≈ 0	1.35

Tabla 7.20: Resultados del AE_D para las instancias de DIMACS

Instancia	Opt o MVC	EA							OCH	CBH
		MVE	Prom(MVE)	#hits	GeM		TiM			
					Min	Prom	Min	Prom		
c-fat-200-1	12	12	12	30	1	1	0	0	12	12
c-fat-200-2	24	24	24	30	1	1	0	0	24	24
c-fat-200-5	58	58	58	30	1	1	0	0	58	58
c-fat-500-1	14	14	14	30	1	1	0	0	14	14
c-fat-500-10	126	126	126	30	1	1	0	0	126	126
c-fat-500-2	26	26	26	30	1	1	0	0	26	26
c-fat-500-5	64	64	64	30	1	1	0	0	64	64
johnson16-2-4	8	8	8	30	1	1	0	0	8	8
johnson32-2-4	16	16	16	30	1	1	0	0	16	16
johnson8-2-4	4	4	4	30	1	1	0	0	4	4
johnson8-4-4	14	14	14	30	1	1	0	0	14	14
p-hat-300-1	8	8	7	7	1	85	0	2.27	8	8
p-hat-300-2	25	25	23	3	1	809	0	2.49	25	25
p-hat-300-3	36	34	31	0	79	1191	0.17	2.65	36	36
p-hat-500-1	9	9	8	7	1	335	0	2.44	9	9
p-hat-500-2	36	34	29	0	155	1243	0.92	7.65	35	36
p-hat-500-3	≥50	48	43	0	212	1742	1.03	8.36	49	49
p-hat-700-1	11	10	9	0	1	581	0	7.74	9	11
p-hat-700-2	44	42	38	0	191	1600	2.06	17.43	44	44
p-hat-700-3	≥62	61	54	0	115	1747	1.29	14.09	62	60
keller4	11	11	11	30	1	70	0	0.9	11	10
keller5	27	27	25	3	1	1205	0	11.96	25	21
san200_0.7_1	30	30	27	23	1	199	0	0.32	30	15
san200_0.7_2	18	18	15	5	1	699	0	1.09	15	12
san200_0.9_1	70	70	70	30	1	76	0	0.1	70	46
san200_0.9_2	60	60	55	17	102	566	0.14	0.79	60	36
san200_0.9_3	44	44	36	2	40	1115	0.06	1.52	36	30
san400_0.5_1	13	13	8	1	1	294	0	1.26	13	8
san400_0.7_1	40	40	25	5	1	225	0	0.86	40	20
san400_0.7_2	30	30	18	1	1	524	0	1.91	30	15
san400_0.7_3	22	22	16	1	1	982	0	3.49	16	14
san400_0.9_1	100	100	84	15	1	1012	0	3.10	100	50
sanr200_0.7	18	18	16	0	1	395	0.20	0.60	18	18
sanr200_0.9	≥42	41	38	0	83	1253	0.25	1.73	42	41
sanr400_0.5	13	12	11	0	1	275	0	1.19	12	12
sanr400_0.7	≥21	21	18	1	1	714	0	2.86	20	20
brock200_1	21	20	18	0	1	333	0	0.79	21	20
brock200_2	12	12	10	2	1	269	0	0.43	11	12
brock200_3	15	15	13	1	1	342	0	0.51	14	14
brock200_4	17	16	15	0	1	518	0	0.91	16	16
brock400_1	27	23	21	0	1	727	0	2.97	24	23
brock400_2	29	23	22	0	1	828	3.72	2.80	24	24
brock400_3	31	23	21	0	1	857	0.41	2.86	24	23
brock400_4	33	24	22	0	1	601	28.67	2.02	24	24
brock800_1	23	19	19	0	1	544	0	6.24	19	20
brock800_2	24	20	18	0	1	803	0	9.13	19	19
brock800_3	25	19	17	0	1	421	0	4.79	19	20
brock800_4	26	19	18	0	1	973	0	11.34	19	19
MANN_a27	126	126	125	3	54	335	0.13	0.76	126	121
MANN_a45	345	343	341	0	70	743	0.49	8.50	343	336

Capítulo 8

Discusión General

8.1. Introducción

Los dos capítulos previos mostraron, a través de un estudio experimental, la aplicabilidad de dos metaheurísticas basadas en una población para la resolución de problemas con restricciones. En cada una de ellas, se incorporó un enfoque de frontera como técnica alternativa para manejo de restricciones en el contexto de problemas continuos y discretos. Los resultados globales de ambos tipos de algoritmos evidenciaron un muy buen desempeño para el conjunto de problemas considerados.

En este capítulo, se consideran algunos aspectos relevantes referidos al comportamiento de los AEs y SHs. Debido a la considerable cantidad de problemas e instancias estudiados en la presente tesis, se han elegido algunos de ellos, los más representativos de lo que se pretende analizar, para mostrar tales aspectos relacionados al desempeño respectivo. Algunos de estos aspectos tienen que ver con el enfoque de frontera en sí y otros, con cada uno de los algoritmos; esto es, aquellos parámetros que determinan el comportamiento general de un algoritmo específico, o aquellos que dependen del dominio de aplicación.

8.2. Convergencia en SH_C y AE_C

La calidad final de los resultados obtenidos por SH_C y AE_C es comparable y competitiva con otros algoritmos específicamente diseñados para resolver el conjunto de problemas G^* descritos en la sección 5.7.1. Algunos de los problemas, como G02, G10 y G13 resultaron de particular interés para ser usados para la verificación de la eficacia del enfoque de frontera dado principalmente por su dificultad y que algunas o todas sus respectivas restricciones se encuentran activas en el óptimo o mejor conocido. También se destacó en los capítulos anteriores, que tanto SH_C como AE_C , no sólo encontraron muy buenas soluciones, sino que además, evidenciaron un comportamiento robusto a lo largo de las 30 ejecuciones para cada uno de los problemas.

En la figura 8.1 se muestra un conjunto de curvas de convergencia obtenidas a partir de los mejores valores objetivos de cada iteración de SH_C y AE_C para los problemas G02, G10 y G13. También se incluye la respectiva curva de convergencia para G01 (un caso más sencillo de resolver).

Lo que se pretende destacar aquí es que ambos algoritmos basados en el enfoque de frontera convergen rápidamente a valores cercanos al óptimo. Principalmente SH_C , el cual muestra un comportamiento similar en todos los problemas. Un caso particular es G10. Si bien la figura 8.1.c muestra una rápida convergencia de SH_C a un valor cercano al óptimo, esta no es una situación que se verifique en todas las corridas, aunque ambos algoritmos logran alcanzar valores muy cercanos al óptimo. En síntesis, las curvas de convergencia son homogéneas para todos los problemas en general y, en particular, para G02 y G13. El único caso en el cual no se verifica esta situación es para G10, donde, en general, las curvas presentan grandes saltos después de entrar en un estado de meseta por varias iteraciones. Esta observación podría ser generalizada para muchos otros algoritmos publicados, los cuales reflejan grandes diferencias

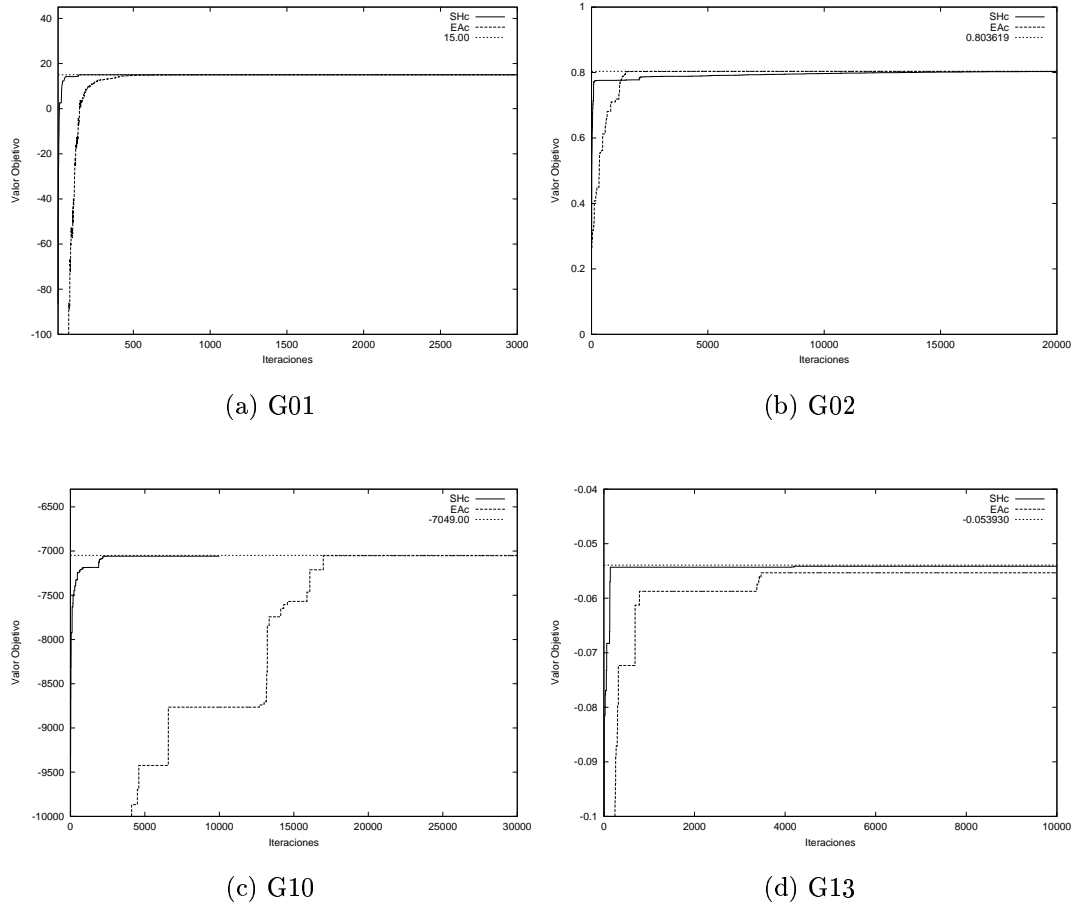


Figura 8.1: Convergencia de AE_C y SH_C para los problemas G01, G02

entre los mejores, medios, peores valores.

8.3. Operadores de frontera *ad hoc* en AE_C

En capítulos anteriores se remarcó que el enfoque de frontera propuesto tenía como antecedente los trabajos de Michalewicz et al. [120, 157] en los cuales se proponen operadores especialmente diseñados para los problemas G02 y G03. Una de las críticas realizadas a esta propuesta apunta a que el método sólo es aplicable si es posible definir operadores que preserven las soluciones generadas sobre la frontera entre el espacio factible y no factible. En el caso particular de G02, una de las dos restricciones de este problema está activa en la mejor solución conocida y sólo

se considera esta restricción para aplicar el operadores especializado¹. El problema G03, por su parte, tiene una única restricción y es del tipo $h(x)$. Para ambos problemas, es posible definir operadores basados en la forma de las ecuaciones que representan la frontera del espacio factible. De esta manera, la frontera de puntos es un conjunto cerrado bajo la aplicación de los operadores de búsqueda. Otro punto a considerar es que el método sólo fue probado con estos problemas y que para cada uno de ellos sólo una restricción se tiene en cuenta, ya sea porque sólo una de ellas está activa en el óptimo o mejor valor conocido (caso G02) o bien, el problema sólo tiene una restricción (caso G03) de igualdad.

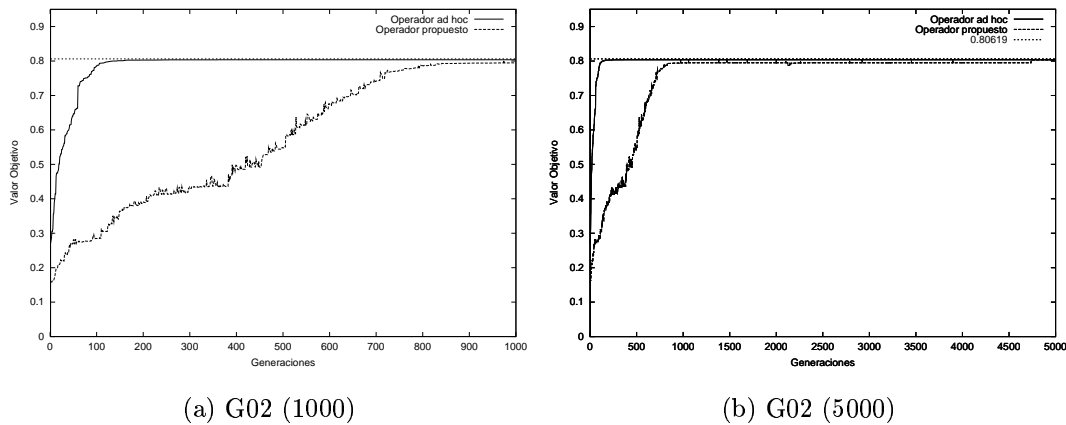


Figura 8.2: Convergencia de AE_C para el problema G02 usando un operador de frontera *ad hoc* y el propuesto en esta tesis. La gráfica de la izquierda muestra el avance hasta la iteración 1000 y la de la derecha hasta la 5000.

A diferencia de lo anterior expuesto, nuestro enfoque de frontera va un paso más allá en dos sentidos. Primero, el método es independiente de la característica que define la restricción sobre la que se quiere realizar la búsqueda. Esto se manifiesta con el hecho que sólo necesita dos puntos, uno de la región factible y otro de la región no factible para proyectar una recta entre ambos para determinar el punto sobre la frontera (ver figura 5.5). Segundo, puede manejar sin mayores complicaciones, problemas con más de una restricción, aunque para ello requiere de alguna

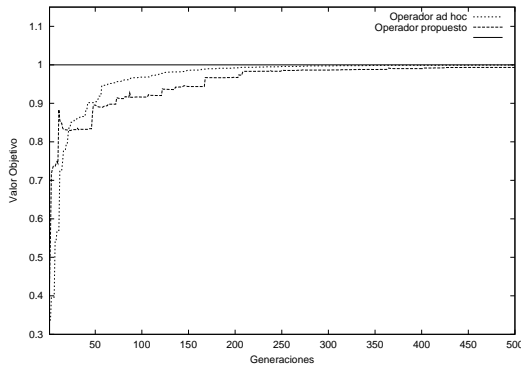
¹En nuestro estudio, como se vio en los capítulos precedentes, optamos por seguir los mismos pasos.

técnica complementaria para manejo de restricciones. Nuestra experiencia mostró que una técnica tan sencilla como la de penalización, fue suficiente para evaluar y verificar el desempeño del enfoque de frontera incorporándolo en dos metaheurísticas basadas en población.

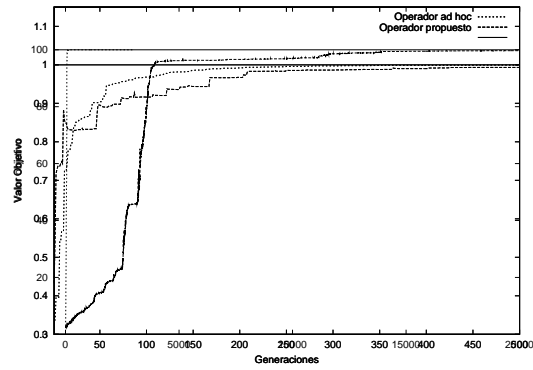
Las siguientes curvas (figuras 8.2 y 8.3) muestran la velocidad de convergencia de AE_C sobre cuatro de los quince problemas estudiados a fin de evaluar el comportamiento de un operador de frontera específicamente diseñado para un tipo de ecuación y otro (nuestra propuesta) con un diseño general.

Los cuatro problemas considerados son G02, G03, G16 y G17. Ellos fueron elegidos dado que existen los operadores de frontera *ad hoc* para G02 y G03. Por otro lado, G16 y G17 tienen ambos una única restricción y es la misma que aparece en la formulación de G03, por lo que es posible elegir el respectivo operador de frontera.

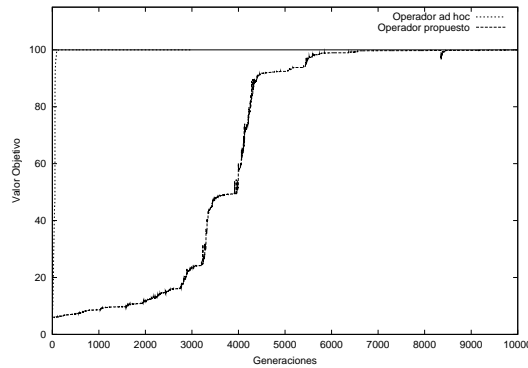
Lo que se muestra en cada una de las gráficas son las curvas de convergencia con respecto a los dos tipos de operadores (*ad hoc* y el generalizado propuesto aquí). Claramente, se ve una mayor velocidad de convergencia para los operadores *ad hoc* sobre todos los problemas, con excepción de G03, para el cual ambos tipos de operadores de frontera convergen de manera similar. A pesar de estas diferencias, ambos tipos de operadores convergen a la mejor solución desde la perspectiva de la calidad final de las soluciones encontradas. Las diferencias de velocidad son razonables cuando se está comparando un método específico contra uno más general. Es decir, lo que se gana en ampliar el espectro de aplicación, se pierde muchas veces en la eficiencia del método. De cualquier manera, el operador de frontera generalizado posee características que lo hacen robusto y potencialmente aplicable a un amplio número de problemas restringidos. Decimos “potencialmente aplicable” debido a que no es posible asegurar que este operador pueda ser usado para cualquier tipo de restricción. Por ejemplo, podría ocurrir que para alguna de ellas, sea difícil ubicar los dos puntos en regiones opuestas para determinar el



(a) G03 (500)



(b) G16 (20000)



(c) G17 (10000)

Figura 8.3: Convergencia de AE_C para los problemas G03, G16 y G17 usando un operador de frontera *ad hoc* y el generalizado. Las mayores diferencias de velocidad de convergencia se aprecian en los problemas G16 y G17.

punto en la frontera. Sin embargo, esta situación no se presentó en el conjunto de problemas considerados.

8.4. Parámetros α y β en SH_D

Es ampliamente reconocido que los parámetros α y β , en un algoritmo ACO, juegan un papel fundamental en su desempeño. Los valores de estos parámetros determinan la importancia relativa que se le asigna a la información local y global en el proceso de construcción de una solución. En otras palabras, ello influyen de manera directa en el balance que el algoritmo puede alcanzar entre la explotación y exploración del espacio

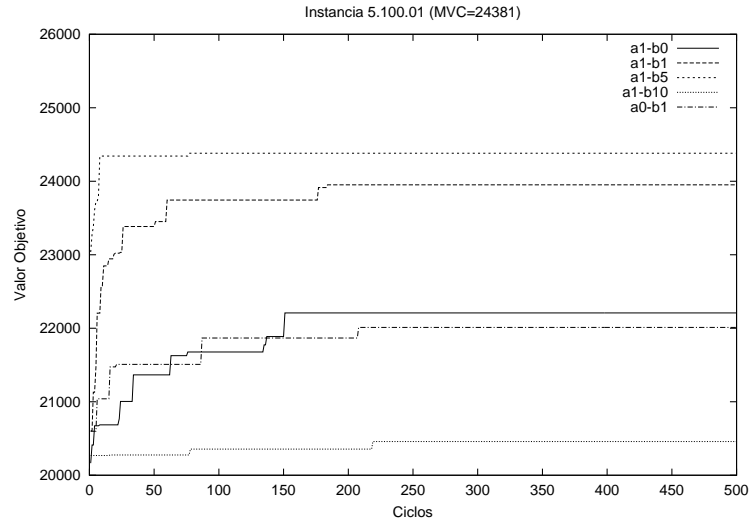


Figura 8.4: Curva de convergencia para una instancia de MKP para distintos valores de α y β .

de búsqueda.

En los experimentos realizados con SH_D (sección 6.3.4) se usaron en general, valores estándares; esto es, $\alpha = 1$ y $\beta = 5$. En la figura 8.4 se muestra la curva de convergencia para la instancia 5.100.01 de MKP (el mejor valor conocido es 24381). En dicha figura se puede apreciar claramente las diferencias entre las curvas de convergencia para distintos valores de estos parámetros. Los valores considerados son $(\alpha, \beta) \in \{(1, 0), (1, 1), (1, 5), (1, 10), (0, 1)\}$. Las primeras cuatro combinaciones de valores se pueden interpretar respectivamente como: sólo se da importancia a la información global (búsqueda guiada exclusivamente por el rastro de feromona); se da igual importancia a la información global y local; mayor importancia a la información local; mucha mayor importancia a la información local y por último, importancia exclusiva a la información local, es decir, búsqueda guiada exclusivamente por la heurística del problema. El algoritmo con esta última combinación puede ser visto como un método greedy-estocástico primitivo con múltiples procesos de búsqueda local.

Puede observarse que la mejor combinación de parámetros es $\alpha = 1$ y $\beta = 5$ (considerada ésta, una combinación estándar de valores). La curva

correspondiente a $\alpha = 1$ y $\beta = 1$ se mantiene muy cercana a la anterior. De hecho, es otra de las combinaciones que dieron buenos resultados para algunos problemas, aunque en general esta combinación requiere de un mayor número de iteraciones ya que el algoritmo es más explorativo.

Darle una importancia excesiva a la información local puede llevar a una convergencia prematura, como se puede apreciar en la curva para $\alpha = 1$ y $\beta = 10$ (la inferior en la figura 8.4). Ubicadas en un punto intermedio, se encuentran las curvas correspondientes a opciones de búsqueda guiadas exclusivamente por información global y local. Estas opciones no son usadas ya que contradicen, y de hecho se refleja en las curvas, la concepción de los algoritmo ACO, es decir, la búsqueda en el espacio de soluciones está guiada por la información local (conocimiento del problema) y global (cooperación entre los agentes).

8.5. Parámetro ρ en SH_D

Los valores de este parámetro determinan los cambios a realizarse sobre la estructura que representa el nivel de rastro de feromona. Dicha estructura representa la memoria global y colectiva de aquellas componentes que son preferibles como candidatas a integrar la solución (en el contexto de los problemas discretos estudiados aquí). Es decir que $0 \leq \rho \leq 1$, el factor de evaporación del rastro (o equivalentemente $1 - \rho$, el factor de persistencia) influye directamente en el proceso de actualización de la memoria global de la colonia. Así, si $\rho = 0$, no se producirá evaporación del rastro acumulado hasta el momento y además se agregará una cantidad adicional acorde a la calidad de la solución. Esto implica mantener una memoria de largo término, aunque puede causar convergencia prematura del algoritmo por acumulación excesiva de rastro sobre algunas componentes. Es decir, el algoritmo tiende a concentrarse en un subconjunto de componentes sin la posibilidad de seguir explorando otras regiones del espacio de búsqueda. En el otro extremo

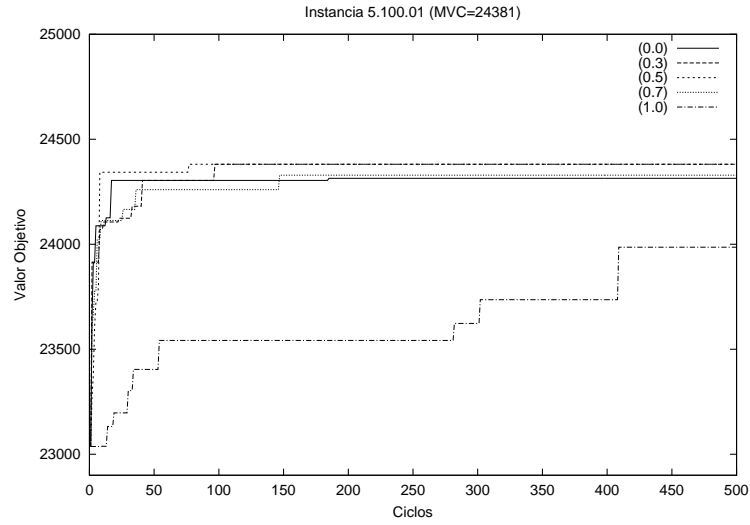
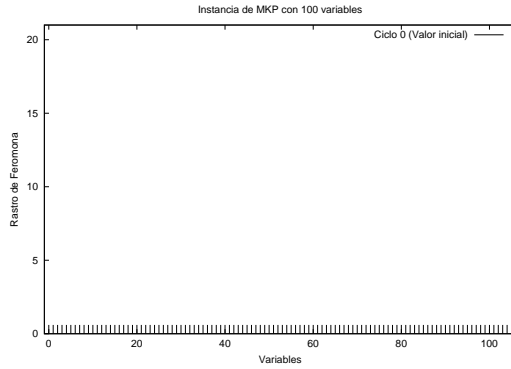


Figura 8.5: Curva de convergencia para una instancia de MKP para distintos valores de la persistencia del rastro ρ con $\alpha = 1$ y $\beta = 5$.

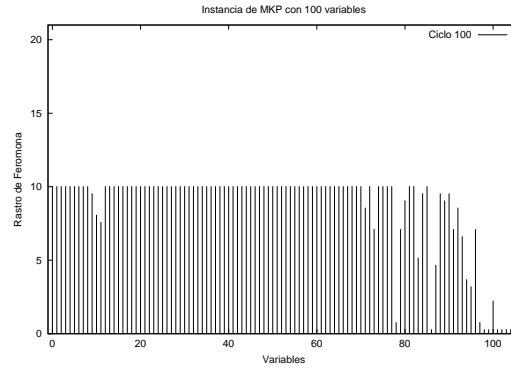
con $\rho = 1$, tampoco deseable, sólo se mantiene memoria a corto plazo, más precisamente, sólo se tiene recuerdo de lo ocurrido en el ciclo previo del algoritmo. Esta forma de actualización, fomenta un comportamiento similar al anterior pero que depende solamente de la calidad de las soluciones encontradas el ciclo inmediato anterior. Los valores intermedios del parámetro ρ establecen un balance entre estas dos situaciones extremas. El valor $\rho = 0,5$ es un valor estándar y es el que fue usado en nuestros experimentos.

la figura 8.5 muestra, para la instancia *5.100.01* de MKP, el efecto en la velocidad de convergencia del algoritmo para distintos valores de ρ . Se puede observar con claridad, que trabajar una memoria de corto plazo (una unidad de tiempo o $\rho = 1$) determina el peor comportamiento del algoritmo. Para el resto de los valores, el algoritmo muestra un desempeño comparable, inclusive con $\rho = 0$. Sin embargo, este valor no es usado dado que, en general, no asegura buenos resultados.

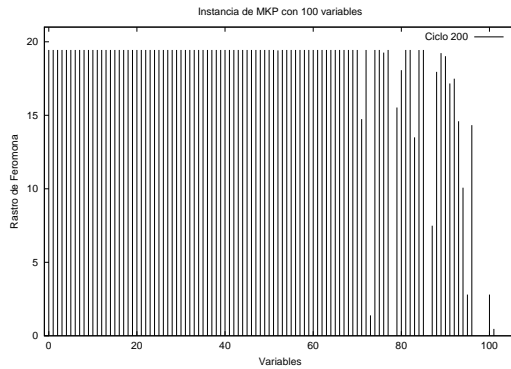
8.6. Rastro de feromona y exploración del espacio de búsqueda



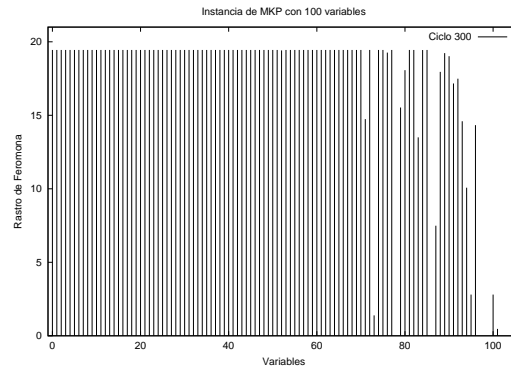
(a) Rastro Inicial



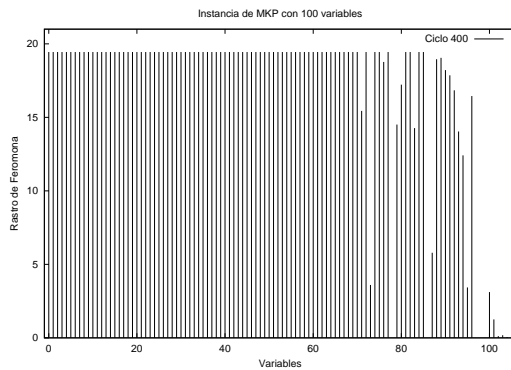
(b) Rastro en el ciclo 100



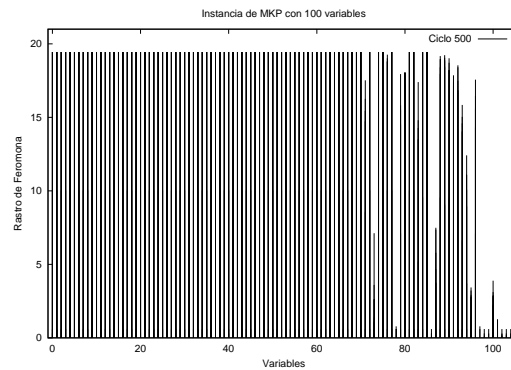
(c) Rastro en el ciclo 200



(d) Rastro en el ciclo 300



(e) Rastro en el ciclo 400



(f) Rastro en el ciclo 500

Figura 8.6: Acumulación del rastro de feromona a través de los ciclos sobre las componentes de una instancia de MKP con 100 variables. Se muestra una instantánea de la acumulación del rastro de feromona cada 100 ciclos.

La visión del comportamiento de un algoritmo ACO para problemas de subconjunto es idéntica a aquella para la cual fue originalmente planteado. Un conjunto de agentes resuelven un problema común en forma independiente y cooperativamente. La diferencia substancial está en la visión del rastro de feromona. Mientras que las primeras versiones de los algoritmos ACO, el rastro se acumula en las *conexiones de las componentes* del problema (asumiendo que el problema tiene tales características), en nuestra visión, el rastro se acumula en las *componentes* del problema dado que aquí se extiende su aplicación a problemas para los cuales la solución es un subconjunto de las componentes del mismo. Es decir, en esta nueva perspectiva se trata de aprender y compartir globalmente la información respecto de la preferencia de incorporar determinadas componentes del problema en las potenciales soluciones. En otras palabras, la extensión de la metaheurística ACO a problemas de subconjunto implica que la estructura original τ que representa el rastro de feromona, una matriz con componentes reales, se transforma en un vector $\tau = (\tau_1, \dots, \tau_n)$, con n , el número de componentes del problema. Tanto la visión original como esta última, un mayor valor para cada entrada en la estructura τ , mayor será la probabilidad de elegir, según corresponda, una conexión o una componente.

Analicemos nuestra visión del rastro de aquí en adelante. Claramente, el vector τ puede ser visto como uno de los *bloques de construcción* de un algoritmo ACO versión subconjunto (por ejemplo, SH_D) dado que sus valores influyen directamente en el cómputo de la probabilidad de selección (línea 9, algoritmo en figura 5.17) de la próxima componente usada en el proceso de construcción de una solución individual.

Supongamos que eliminamos el valor aportado por la heurística del problema. La manera en que se calcula la probabilidad de selección de una componente dependerá exclusivamente de los valores de la memoria global τ , esto es:

La solución anterior excluye justamente las componentes asociadas a los bits más significativos. Intuitivamente, si solamente usáramos el valor de τ (figura 8.6.e) para construir una solución paso a paso adoptando la distribución de probabilidades de la ecuación 8.1, las componentes que deberían ser parte de la solución tendrán mayor probabilidad de ser seleccionadas, siendo estos valores de probabilidades similares entre sí.

Esta observación, abre una perspectiva de análisis teóricos muy interesantes con respecto al comportamiento de estos algoritmos y la posibilidad de explotar propiedades inherentes a los mismos. También es importante notar las similitudes, tal como fue observado por Bonabeau et al. [26], entre los algoritmos ACO y la clase de algoritmos evolutivos con aprendizaje incremental basado en población (PBIL²). Una descripción detallada de este enfoque puede ser encontrada en [13]. Más precisamente, en un algoritmo ACO (para problemas de subconjunto) existe una conexión muy cercana entre el *vector generador* según la denominación en el enfoque PBIL y la información del rastro, la cual es representada por un vector de números reales (el bloque de construcción de este enfoque). Sin embargo, un algoritmo ACO, además de la información del rastro, también usa información heurística para dirigir la búsqueda, a diferencia de PBIL que sólo usa la información de dicho vector generador.

La visión del rastro en la implementación para problemas continuos es diferente de lo anteriormente analizado. En este caso, el rastro de feromona se deposita sobre cada una de las direcciones (o punto de referencia) dispersas en el espacio de búsqueda. Así, τ_i indica la preferencia de realizar una búsqueda en un radio cercano al punto de referencia ubicado en la dirección i . Los valores de τ variarán de acuerdo a la calidad relativa de las soluciones encontradas en cada una de las direcciones. Por otro lado, los puntos iniciales varían de una ejecución a otra, lo cual es un indicador adicional de que τ no está relacionado con el concepto de bloque de construcción y las respectivas soluciones como en el caso de

²Population-Based Incremental Learning.

los problemas de subconjunto. Aquí se usa para distribuir el conjunto de agentes a través de las sucesivas iteraciones en función de la calidad de las soluciones encontradas en cada dirección y lograr una explotación de las mismas.

8.7. Observaciones generales

Es claro que el análisis realizado previamente está sesgado hacia el comportamiento de los algoritmos ACO implementados en este estudio. Esto se justifica desde la siguiente perspectiva. Los AEs, la importancia de sus componentes y los efectos de los valores de sus parámetros, han sido estudiados ampliamente. Sin embargo, no ocurre lo mismo con los algoritmos ACO y en especial con el algoritmo propuesto aquí para problemas restringidos de subconjunto y continuos.

Una observación global con respecto al enfoque de frontera propuesto, es que pudo ser incorporado y probado fácilmente en las dos metaheurísticas consideradas. En el caso de los problemas continuos (más precisamente NLP) el enfoque de frontera propuesto es idéntico respecto de la representación y mecanismo complementario para el manejo de restricciones (cuando es necesario incorporarlo). La diferencia de cada método, ya sea AE o ACO, está en la forma que cada uno guía la búsqueda hacia las regiones más promisorias del espacio de soluciones.

En cuanto a los problemas discretos, se pueden establecer mayores diferencias entre AEs y algoritmos ACO con respecto a la manera en que exploran el espacio de búsqueda. Nuevamente aquí entran en juego los bloques de construcción de cada algoritmo. En un algoritmo ACO, cada agente construye paso a paso una solución basándose en información global (τ) y local (η). Los cambios realizados sobre τ influyen directamente en el cómputo de las probabilidades de selección de cada componente. En el caso de los AEs, su bloque de construcción son los esquemas contenidos en las soluciones. La recombinación de soluciones de buena

calidad (vía los operadores de frontera definidos aquí) generan soluciones que heredan ciertos esquemas comunes a los padres. La información no heredada es completada paso a paso, hasta que la solución alcance al frontera.

Capítulo 9

Conclusiones y Trabajos Futuros

En esta tesis se propuso un enfoque de frontera como estrategia de búsqueda acotada a una región reducida del espacio de soluciones para resolver problemas de optimización restringidos.

El enfoque propuesto fue incorporado como técnica alternativa para manejo de restricciones en dos metaheurísticas diferentes. Dichas metaheurísticas son: algoritmos evolutivos y algoritmos basados en el comportamiento de las colonias de hormigas.

El enfoque de frontera fue concebido desde dos perspectivas distintas, para problemas de optimización combinatoria y para problemas de programación no lineal. Su incorporación en las metaheurísticas consideradas implicó realizar distintos tipos de adaptaciones en cada una de ellas. Los AEs son los que exigieron menos requerimientos de adaptación en sus componentes, ya sea para el caso discreto o continuo. Para los algoritmos ACO (caso discreto) se propuso una extensión al método original para manejar el tipo de problemas considerados en esta tesis: MKP, SCP y MISP. Problemas a los que hemos categorizado como *problemas de subconjunto*. Para el caso continuo, el algoritmo ACO implementado, se basa en en propuestas anteriores para discretizar el espacio de búsqueda, aunque aquí se lo extiende de manera tal que incorpora el manejo de restricciones a través del enfoque de frontera.

A continuación, se presentan las observaciones más relevantes respecto

del trabajo realizado:

- Se mostró a través de un estudio experimental, la aplicabilidad del enfoque de frontera propuesto. El estudio experimental involucró un conjunto considerable y representativo de los tipos de problemas a los que se pretende abordar a través del enfoque de frontera.
- Los resultados mostrados por los AEs y SHs implementados, son en su gran mayoría, comparables a los obtenidos por otros métodos y/o algoritmos. En el caso particular de los problemas NLP, el enfoque mostró ser efectivo como técnica alternativa para el manejo de restricciones y al mismo tiempo, implementable a través de distintas metaheurísticas.

Para los problemas discretos, se verificaron ciertas diferencias en su efectividad para determinadas instancias de los problemas considerados. Sin embargo, el enfoque de frontera implementado a través de SH_D y AE_D , permitió resolver óptimamente muchas de las instancias consideradas para cada uno de los problemas. Se obtuvieron además, resultados comparables a otros algoritmos, al menos desde el punto de vista de la calidad final de las soluciones alcanzadas.

- En términos comparativos y para ambos dominios de aplicación, los AEs y SHs lograron resultados similares en la mayoría de los casos. Si bien lo anterior no es suficiente para afirmar que se debe al enfoque de frontera implementado en ambas metaheurísticas, al menos refuerza dicha idea.
- Para los problemas discretos y para ambas metaheurísticas, el enfoque de frontera se implementó sin necesidad de incorporar penalización o representaciones avanzadas. De hecho, sólo fue necesario implementar un control mínimo sobre las soluciones en el proceso de generación (recordar la manera en que se construyen las soluciones

en el algoritmo SH_D y en que se aplican los operadores genéticos en AE_D).

Para los problemas continuos, se requiere (al menos para aquellos problemas con más de una restricción) la incorporación de algún mecanismo complementario para el manejo de restricciones. Cabe aclarar aquí, que el enfoque propuesto es lo suficientemente flexible para aceptar distintos mecanismos que eventualmente mejoren su desempeño o amplíen el espectro de problemas a los que se puede aplicar exitosamente. Sin embargo, una técnica de penalización simple fue suficiente para resolver los problemas considerados (o alcanzar valores muy cercanos al óptimo).

- Una de las desventajas ampliamente reconocidas de las metaheurísticas, en general, es su elevado número de parámetros y por ende, el trabajo requerido para ajustar correctamente los valores de dichos parámetros. Si a esta situación, se le agrega el hecho que la técnica para manejo de restricciones incorporada en la metaheurística, incluye parámetros adicionales, la desventaja mencionada anteriormente se profundiza aún más.

Desde el punto de vista de los problemas discretos, el enfoque de frontera no añade parámetros adicionales a las metaheurísticas, dado que los AEs y SHs toman información del problema cuando requieren tomar decisiones en el proceso de generación de las soluciones. Sin embargo, para los problemas continuos, surgen parámetros adicionales: un parámetro para indicar la restricción sobre la cual realizar la búsqueda; si la búsqueda se realizará en forma cíclica sobre “Todas” las restricciones o sólo las “Activas” y en estos casos, cada cuántas iteraciones realizar el cambio. Otro parámetro, para nuestra implementación, es si la penalidad es dinámica o estática; en ambos casos, se necesita de un valor de entrada adicional.

En función de las observaciones anteriores, se proponen aquí algunas

posibles extensiones y líneas de investigación para la temática abordada en esta tesis:

- Eliminar, para el caso continuo, los parámetros propios del enfoque de frontera propuesto. Por un lado, se podría eliminar aquellos parámetros que tienen que ver con la decisión de la restricción a considerar en el proceso de búsqueda en un tiempo dado. Esto podría realizarse a través de una forma de aprendizaje automático en relación al cambio o permanencia sobre una restricción en particular, según la calidad y/o grado de violación de ciertas restricciones. Asimismo, se pueden considerar técnicas de manejo de restricciones que no incluyan parámetros. Esto implicaría reducir aún más los parámetros potencialmente necesarios para la aplicación del enfoque de frontera en el dominio continuo.
- Intentar aislar el concepto subyacente al enfoque de frontera propuesto, independientemente de la metaheurística sobre la cual está implementado. De esta manera se podrían derivar propiedades inherentes al enfoque de frontera que determinen su aplicabilidad, independientemente de la metaheurística o algoritmo que explote dicho concepto. Lo que puede cambiar de un método a otro, es claramente la manera que cada uno explora o explota el espacio de búsqueda, pero todos ellos se basan en el mismo principio: “restringir la búsqueda por la frontera entre la región factible y no factible del espacio de soluciones” o “sesgar la búsqueda hacia la frontera la región factible y no factible del espacio de soluciones”.
- En el caso de los algoritmos ACO en general y la nueva versión propuesta aquí en particular, requieren de un análisis teórico en profundidad con respecto a las propiedades de convergencia y características inherentes a su comportamiento. Recientemente han surgido algunos estudios de la comunidad de algoritmos ACO enfocados hacia la justificación formal de su desempeño y derivaciones

de algunas propiedades de convergencia. Por cierto, la derivación de ciertas propiedades demostrables formalmente, permitirían mejorar algunos aspectos de su funcionamiento y/o brindar la posibilidad de crear nuevos conceptos de búsqueda.

- Por último, aunque no menos importante, se destacan nuevas aplicaciones de SH_D a problemas de subconjunto. En este caso, nos referimos al problema de scheduling *minimum tardy task problem* para el cual fue desarrollado y exitosamente implementado un algoritmo ACO para su resolución [5].

Por otro lado, cabe destacar la pertinencia del estudio en este campo dado el actual interés en el desarrollo de nuevos métodos y técnicas para manejo de restricciones, siendo ésta un área muy activa de investigación. Basta mencionar algunos de los últimos trabajos presentados en el último congreso de computación evolutiva (CEC04): uno de ellos, Toscano et al. [136], aborda los problemas G^* a través del método llamado Particle Swarm Optimization mientras que el otro, propuesto por Yuchi et al. [178] aborda dichos problemas desde la perspectiva de los algoritmos evolutivos. También cabe mencionar que los resultados encontrados por SH_C y AE_C son competitivos con los últimos enfoques destinados a la resolución de este tipo de problemas con restricciones.

Bibliografía

- [1] E. Aarts and J. Lenstra. *Local search in combinatorial optimization*. John Wiley & Sons, Chichester, 1997.
- [2] Emile H. L. Aarts and Jan Korst. *Simulated Annealing and Boltzmann Machines*. Wiley, Chichester, 1989.
- [3] C. Aggarwal, J. Orlin, and R. Tai. Optimized Crossover for the Independent Set Problem. *Operations Research*, 45(2):226–234, 1997.
- [4] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *Data Structures and Algorithms*. Addison-Wesley Series in Computer Science and Information Processing. Addison-Wesley, 1982. AHO a 82:1 P-Ex.
- [5] E. Alba, G. Leguizamón, and G. Ordoñez. Evolutionary algorithms for the minimum tardy task problem. In *Proceedings of the International Conference on Computer Science, Software Engineering, Information Technology, e-Business, and Applications*, pages 401–413, Rio de Janeiro, Brasil, 2003.
- [6] A. Newell, J. C. Shaw, and H. A. Simon. The processes of creative thinking. In H. E. Gruber, G. Terrell, and M. Wertheimer, editors, *Contemporary approaches to creative thinking*, pages 63–119. Atherton Press, New York, 1962.
- [7] T. Bäck. GENEsYs, software basado en el paquete GENESIS propuesto por Grefenstette. <http://www.aic.nrl.navy.mil/galist/src/>.

- [8] T. Bäck, D. B. Fogel, and Z. Michalewicz, editors. Oxford University Press, New York, and Institute of Physics Publishing, Bristol, 1997.
- [9] T. Bäck, D.B. Fogel, and Z. Michalewicz, editors. *Evolutionary Computation 1 (Basic Algorithms and Operators)*. Institute of Physics Publishing, Bristol and Philadelphia, 2000.
- [10] T. Bäck, D.B. Fogel, and Z. Michalewicz, editors. *Evolutionary Computation 2 (Advanced Algorithms and Operators)*. Institute of Physics Publishing, Bristol and Philadelphia, 2000.
- [11] T. Bäck, F. Hoffmeister, and H. Schwefel. A survey of evolution strategies. In *Fourth International Conference on Genetic Algorithms*. Morgan Kaufmann, San Mateo, CA., 1991.
- [12] T. Bäck and S. Khuri. An Evolutionary Heuristic for the Maximum Independent Set Problem. In M. Michalewicz et. al., editor, *Proceedings of the First International Conference on Evolutionary Computation*, pages 531–535, Piscataway, NJ, 1994. IEEE Press.
- [13] S. Baluja. Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Technical Report Tech. Rep. No. CMU-CS-94-163, Carnegie Mellon University, Pittsburgh, PA, 1994.
- [14] S. Baluja. An empirical comparison of seven iterative and evolutionary function optimization heuristics. Technical Report CMU-CS-95-193, School of Computer Science, Carnegie Mellon University, 1995.
- [15] R.S. Barr, B.L. Golden, J.P. Kelly, R. Resende, and W.R. Stewart. Designing and reporting on computational experiments with heuristic methods. *Journal of Heuristics*, 1(1):9–32, 1995.

- [16] R. Battiti and G. Tecchiolli. The reactive tabu search. *ORSA Journal on Computing*, 6:126–140, 1994.
- [17] J. Bean and A. Hadj-Alouane. A dual genetic algorithm for bounded integer programs. Technical Report TR 92-53, Department of Industrial and Operations Engineering, The University of Michigan, 1992. To appear in R.A.I.R.O.-R.O. (invited submission to special issue on GAs and OR).
- [18] J. Beasley and P. C. Chu. A genetic algorithm for the set covering problem. *European Journal of Operational Research*, 94(2):393–404, 1996.
- [19] J. Beasley and P. C. Chu. A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics*, 4:63–86, 1998.
- [20] J. E. Beasley. OR-Library: Distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072, 1990.
- [21] G. Bilchev and I. Parmee. Ant colony search vs. genetic algorithms. Technical report, Plymouth Engineering Design Centre, University of Plymouth, 1995.
- [22] G. Bilchev and I. C. Parmee. The ant colony metaphor for searching continuous design spaces. *Lecture Notes in Computer Science*, 993:25–39, 1995.
- [23] G. Bilchev and I.C. Parmee. Constrained and Multi-Modal Optimisation with an Ant Colony Search Model. In Ian C. Parmee and M. J. Denham, editors, *Proceedings of 2nd International Conference on Adaptive Computing in Engineering Design and Control*. University of Plymouth, Plymouth, UK, March 1996.
- [24] G. Bilchev and I.C. Parmee. Constrained and Multi-Modal Optimisation with an Ant Colony Search Model. In Ian C. Parmee and

- M. J. Denham, editors, *Proceedings of 2nd International Conference on Adaptive Computing in Engineering Design and Control*. University of Plymouth, Plymouth, UK, March 1996.
- [25] B. Bollobás. *Random Graphs*. Academic Press, 1985.
- [26] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence - From Natural to Artificial Systems*. Santa Fe Institute Studies in the Sciences of Complexity. Oxford University Press, 1999.
- [27] E. Bonabeau, M. Dorigo, and G. Theraulaz. Inspiration for optimization from insect behaviour. *Nature*, Vol 206, pages 39-42, July 2000.
- [28] H. J. Bremermann. The evolution of intelligence: The nervous system as a model of its environment. Technical Report 1, Contract No. 477(17), Department of Mathematics, University of Washington, Seattle, July 1958.
- [29] H. J. Bremermann. Optimization through evolution and recombination. In M. C. Yovits et al., editor, *Self-Organizing Systems*. Spartan Books, Washington, D.C., 1962.
- [30] B. Bullnheimer, R.F. Hartl, and C. Strauss. An new rank based version of the ant system: A computational study. In *Working paper # 1, SFB Adaptive Information Systems and Modelling in Economics and Management Science*.
- [31] R. Landa Becerra C. A. Coello Coello A1. A cultural algorithm for constrained optimization. In L.E. Sucar O.C. Battistutti (Eds.) C.A. Coello Coello, A. de Albornoz, editor, *MICAI 2002: Advances in Artificial Intelligence : Second Mexican International Conference on Artificial Intelligence*, volume 2313 of *Lecture Notes in Computer Science*, page 98. Springer-Verlag Heidelberg, Yucatan, Mexico, 2002.

- [32] P. Chu. *A Genetic Algorithm Approach for Combinatorial Optimization Problems*. PhD thesis, The Management School, Imperial College of Science, Technology and Medicine, London, 1997.
- [33] P. Chu and J. Beasley. A genetic algorithm for the set partitioning problem. Technical report, Imperial College, The Management School, London, England, 1995.
- [34] Carlos A. Coello Coello and Efrén Mezura-Montes. Handling Constraints in Genetic Algorithms Using Dominance-Based Tournaments. In I.C. Parmee, editor, *Proceedings of the Fifth International Conference on Adaptive Computing Design and Manufacture (ACDM 2002)*, volume 5, pages 273–284, University of Exeter, Devon, UK, April 2002. Springer-Verlag.
- [35] Carlos A. Coello Coello. Constraint-handling using an evolutionary multiobjective optimization technique. *Civil Engineering and Environmental Systems*, 17:319–346, 2000.
- [36] Carlos A. Coello Coello. Treating Constraints as Objectives for Single-Objective Evolutionary Optimization. *Engineering Optimization*, 32(3):275–308, 2000.
- [37] Carlos A. Coello Coello and Ricardo Landa Becerra. Adding Knowledge and Efficient Data Structures to Evolutionary Programming: A Cultural Algorithm for Constrained Optimization. In W.B. Langdon, E.Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A.C. Schultz, J. F. Miller, E. Burke, and N.Jonoska, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2002)*, pages 201–209, San Francisco, California, July 2002. Morgan Kaufmann Publishers.

- [38] J.F. Cordeau, M. Gendreau, and G. Laporte. A tabu search heuristic for periodic and multi-depot vehicle routing problems. *Networks: An International Journal*, 30:105–119, 1997.
- [39] D. Corne, M. Dorigo, and F. Glover, editors. *New Ideas in Optimization*. McGraw-Hill International, 1999.
- [40] E. de San Pedro, D. Pandolfi, A. Villagra, M. Lasso, and R. Gallard. Effect of crossover operators under multirecombination: Weighted tardiness, a test case. In *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, pages 699–705, Portland, Oregon, 20-23 June 2004. IEEE Press.
- [41] K. Deb. An Efficient Constraint Handling Method for Genetic Algorithms. *Computer Methods in Applied Mechanics and Engineering*, 186(2/4):311–338, 2000.
- [42] V. Dhar and N. Ranganathan. Integer programming vs. expert systems: An experimental comparison. *Communications of the ACM*, 33:338–348, 1990.
- [43] DIMACS directories. Second DIMACS implementation challenge. <http://dimacs.rutgers.edu/Challenges/index.html>.
- [44] M. Dorigo and L.M. Gambardella. A study of some properties of ant-q. In *Fourth International Conference on Parallel Problem Solving from Nature, PPSN IV*, pages 656–665, Berlin, 1996. Springer-Verlag.
- [45] M. Dorigo and L.M. Gambardella. Ant colonies for the traveling salesman problem. *Biosystems*, 47:73–81, 1997.
- [46] M. Dorigo and L.M. Gambardella. Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.

- [47] M. Dorigo, V. Maniezzo, and A. Coloni. Positive feedback as a search strategy. Technical Report 91-016, Politecnico di Milano, 1991.
- [48] M. Dorigo, V. Maniezzo, and A. Coloni. Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics (Part B)*, 26(1):29–41, 1996.
- [49] M. Dorigo and T. Stützle. *Ant Colony Optimization*. Mit-Press, 2004.
- [50] Marco Dorigo, Vittorio Maniezzo, and Alberto Coloni. Ant system: Optimization by a colony of cooperating agents. *IEEE Trans. on Systems, Man, and Cybernetics—Part B*, 26(1):29–41, 1996.
- [51] J. Dréo and P. Siarry. A new ant colony algorithm using the hierarchical concept aimed at optimization of multimodal continuous functions. In M. Dorigo et al., editor, *ANTS*, pages 216–221, Heidelberg, 2002. Springer-Verlag Berlin.
- [52] A. E. Eiben, P.-E. Raue, and Z. Ruttkay. Genetic algorithms with multi-parent recombination. *Lecture Notes in Computer Science*, 866:78–84, 1994.
- [53] A. E. Eiben, C. H. M. Van Kemenade, and J.N. Kok. Orgy in the computer: Multi-parent reproduction in genetic algorithms. *Lecture Notes in Computer Science*, 929:934–939, 1995.
- [54] S. Esquivel, C. Gatica, and R. Gallard. Conventional and multirecombinative evolutionary algorithms for the parallel task scheduling problem. *Lecture Notes in Computer Science*, 2037:223–228, 2001.
- [55] S. C. Esquivel, A. Leiva, and R. H. Gallard. Multiple crossover per couple in genetic algorithms. In *1997 IEEE International Confe-*

- rence on Evolutionary Computation (ICEC '97)*, pages 103–106, University Place Hotel, Indianapolis, USA, 1997.
- [56] Susana C. Esquivel, Héctor A. Leiva, and Raúl H. Gallard. Multiple crossovers between multiple parents to improve search in evolutionary algorithms. In *1999 Congress on Evolutionary Computation*, pages 1589–1594, Piscataway, NJ, 1999. IEEE Service Center.
- [57] L Ferguson and W E Hart. A filter-based evolutionary algorithm for constrained optimization (extended abstract). pages 287–290, 2003.
- [58] Christodoulos A. Floudas and P. M. Pardalos. *A collection of test problems for constrained global optimization algorithms*, volume 455. Springer-Verlag Inc., New York, NY, USA, 1990.
- [59] L. J. Fogel. Autonomous automata. *Industrial Research*, 4:14–19, 1962.
- [60] L. J. Fogel. Decision making by automata. Technical Report GDA-ERR-AN-222, General Dynamics, San Diego, CA, 1962.
- [61] L. J. Fogel. Toward inductive inference automata. In *Proceedings of the International Federation for Information Processing Congress*, pages 395–399, Munich, 1962.
- [62] L. J. Fogel. *On the Organization of Intellect*. PhD thesis, University of California, Los Angeles, CA, 1964.
- [63] C. Fonlupt, D. Robilliard, and O. Roux. Co-operative improvement for a combinatorial optimization algorithm. *Artificial Evolution*, pages 231–241, 1999.
- [64] R. M. Friedberg. A learning machine: Part I. *IBM Journal*, 2(1):2–13, January 1958.

- [65] R. M. Friedberg, B. Dunham, and J. H. North. A learning machine: Part II. *IBM Journal*, 3(7):282–287, July 1959.
- [66] M. Schütz G. Leguizamón, Z. Michalewicz. An ant system for the maximum independent set problem. In *Proceedings of the 2001 Argentinian Congress on Computer Science*, pages 1027–1040, El Calafate, Argentina,, 2001.
- [67] G. Nemhauser and L. Wolsey. *Integer and Combinatorial Optimization*. Wiley Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, 1988.
- [68] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.
- [69] M. Gen, K. Ida, and J. Kim. A spanning tree-based genetic algorithm for bicriteria topological network design. In *IEEE International Conference Evolutionary Computation. IEEE World Congress on Computational Intelligence*, pages 15–20, 1998.
- [70] L.E. Gibbons, D.W. Hearn, and P.M. Pardalos. A continuous Based Heuristic for the Maximum Clique Problem. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 21:103–124, 1996.
- [71] F. Glover. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8:156–166, 1977.
- [72] F. Glover. Tabu search fundamentals and uses. Technical report, University of Colorado, Boulder, 1994.
- [73] F. Glover and G.A.Kochenberg, editors. *Handbook of Metaheuristics*. Kuwler Academic Publishers, London, 2003.

- [74] F. Glover and G. Kochenberger. *Critical events tabu search for multidimensional knapsack problem*. In *Metaheuristics: theory and applications*, pages 407–428. Kluwer Academic Publishers.
- [75] F.W. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.
- [76] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Publishing Company, Inc., Reading, MA, 1989.
- [77] S. Goss, S. Aron, J.L. Deneubourg, and J.M. Pasteels. Self-organized shortcuts in the argentine ant. *Naturwissenschaften*, 76:579–581, 1989.
- [78] J. Gottlieb. On the Feasibility Problem of Penalty-Based Evolutionary Algorithms for Knapsack Problems. In E.J.W. Boers, S. Cagnoni, J. Gottlieb, E. Hart, P.L. Lanzi, G. Raidl, R.E. Smith, and H. Tijink, editors, *Proceedings of EvoWorkshops Lecture Notes in Computer Science*, volume 2037, pages 50–59. Springer, 2001.
- [79] J. Gottlieb and G.R. Raidl. Characterizing Locality in Decoder-Based EAs for the Multidimensional Knapsack Problem. In Cyril Fonlupt, Jin-Kao Hao, Evelyne Lutton, Edmund Ronald, and Marc Schoenauer, editors, *Proceedings of the 4th European Conference on Artificial Evolution (AE 1999)*, pages 38–52, Heidelberg, Germany, November 1999. Dunkerque, France, Springer-Verlag. Lecture Notes in Computer Science Vol. 1829.
- [80] Jens Gottlieb. On the Effectivity of Evolutionary Algorithms for the Multidimensional Knapsack Problem. In Cyril Fonlupt, Jin-Kao Hao, Evelyne Lutton, Edmund Ronald, and Marc Schoenauer, editors, *Proceedings of the 4th European Conference on Artificial Evolution (AE 1999)*, pages 23–37, Heidelberg, Germany, Novem-

- ber 1999. Dunkerque, France, Springer-Verlag. Lecture Notes in Computer Science Vol. 1829.
- [81] Jens Gottlieb. On the Feasibility Problem of Penalty-Based Evolutionary Algorithms for Knapsack Problems. In E.J.W. Boers, S. Cagnoni, J. Gottlieb, E. Hart, P.L. Lanzi, G. Raidl, R.E. Smith, and H. Tijink, editors, *Proceedings of EvoWorkshops Lecture Notes in Computer Science*, volume 2037, pages 50–59. Springer, 2001.
- [82] S. Ben Hamida and Marc Schoenauer. An Adaptive Algorithm for Constrained Optimization Problems. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J.J. Merelo, and H.-P. Schwefel, editors, *Proceedings of 6th Parallel Problem Solving From Nature (PPSN VI)*, pages 529–538, Heidelberg, Germany, September 2000. Paris, France, Springer-Verlag. Lecture Notes in Computer Science Vol. 1917.
- [83] Sana Ben Hamida and Marc Schoenauer. ASCHEA: New Results Using Adaptive Segregational Constraint Handling. In *Proceedings of the Congress on Evolutionary Computation 2002 (CEC'2002)*, volume 1, pages 884–889, Piscataway, New Jersey, May 2002. IEEE Service Center.
- [84] Michael Pilegaard Hansen. Solving multiobjective knapsack problems using MOTS, July 11 (<http://www.imm.dtu.dk/mph/papers/mic97.ps>) 1997.
- [85] Michael Pilegaard Hansen. Tabu Search in Multiobjective Optimisation : MOTS. In *Proceedings of the 13th International Conference on Multiple Criteria Decision Making (MCDM'97)*, Cape Town, South Africa, 1997.
- [86] D. Himmelblau. *Applied Nonlinear Programming*. McGraw-Hill, New York, 1972.

- [87] Robert Hinterding and Zbigniew Michalewicz. Your Brains and My Beauty: Parent Matching for Constrained Optimisation. In *Proceedings of the 5th International Conference on Evolutionary Computation*, pages 810–815, Anchorage, Alaska, May 1998.
- [88] J. H. Holland. *Adaptation in natural and artificial systems*. The University of Michigan Press, Ann Arbor, MI, 1975.
- [89] A. Homaifar, C. Qi, and S. Lai. Constrained optimization via genetic algorithms. *Simulation*, 62(4):242–254, 1994.
- [90] Wen-Chih Huang, Cheng-Yan Kao, and Jorng-Tzong Horng. A genetic algorithm approach for set covering problems. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, pages 569–574. IEEE Press, 1994.
- [91] J. Nocedal and S.J. Wright. *Numerical Optimization*. Springer, 1999.
- [92] Jeffrey D. Ullman (Contributor) John E. Hopcroft. *Introduction to Automata Theory, Languages, and Computation*. 1979.
- [93] J. Joines and C. Houck. The use of nonstationary penalty functions to solve nonlinear constrained optimization problems with GAs. In *International Conference on Evolutionary Computation*, pages 579–584. IEEE Press, Piscataway, 1994.
- [94] K. A. De Jong. *An analysis of the behaviour of a class of genetic adaptive systems*. PhD thesis, University of Michigan, 1975. Diss. Abstr. Int. 36(10), 5140B, University Microfilms No. 76–9381.
- [95] S. A. Kazarlis, S. E. Papadakis, J. B. Theoharis, and V. Petridis. Microgenetic Algorithms as Generalized Hill Climbing Operators for GA Optimization. *IEEE Transactions on Evolutionary Computation*, 5(3):204–217, June 2001.

- [96] Andy Keane. Genetic algorithms digest, v8n16, 1994.
- [97] J. Kennedy and R Eberhart. Particle swarm optimization. In *IEEE International Conference on Neural Networks*, volume IV, pages 1942–1948, Perth, Australia. IEEE Service Center.
- [98] S. Kirkpatrick, C. D. Gelatt (Jr.), and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, May 1983.
- [99] J. R. Koza. Hierarchical genetic algorithms operating on populations of computer programs. pages 768–774. Morgan Kaufmann Publishers, San Mateo, CA, 1989.
- [100] J. R. Koza. Evolution and co-evolution of computer programs to control independently-acting agents. In Jean-Arcady Meyer and Stewart W. Wilson, editors, *From Animals to Animats. Proceedings of the 1st International Conference on Simulation of Adaptive Behavior*, pages 366–375, Cambridge, MA, 1991. MIT Press.
- [101] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Complex Adaptive Systems. The MIT Press, Cambridge, MA, 1992.
- [102] J. R. Koza. The genetic programming paradigm: Genetically breeding populations of computer programs to solve problems. In Brancko Souček and the IRIS Group, editors, *Dynamic, Genetic, and Chaotic Programming*, chapter 10, pages 203–321. Wiley, 1992.
- [103] S. Koziel and Z. Michalewicz. A decoder-based evolutionary algorithm for constrained parameter optimization problems. In *5th Conference on Parallel Problems Solving from Nature*. Springer Verlag, 1998.
- [104] Slawomir Koziel and Zbigniew Michalewicz. Evolutionary Algorithms, Homomorphous Mappings, and Constrained Parameter Optimization. *Evolutionary Computation*, 7(1):19–44, 1999.

- [105] C.H. Lee, K.P. Park, and J.H. Kim. Hybrid Parallel Evolutionary Algorithms for constrained optimization utilizing PC Clustering. In *Proceedings of the Congress on Evolutionary Computation 2001 (CEC'2001)*, volume 2, pages 1436–1441, Piscataway, New Jersey, May 2001. IEEE Service Center.
- [106] G. Leguizamón and Z. Michalewicz. A New Version of Ant System for Subset Problems. In *Proceedings of the 1999 Congress on Evolutionary Computation*, pages 1459–1464. IEEE Press, Piscataway, NJ, 1999.
- [107] Guillermo Leguizamón. The ant colony metaphor in continuous spaces using boundary search. In *Proceedings of the XI Congreso Argentino de Ciencia de la Computación*, pages pp. 740–751, La Plata, Argentina, Octubre 2003.
- [108] W. Lei and W. Qidi. Ant system algorithm for optimization in continuous space. In *proceedings of the 2001 IEEE International Conference on Control Applications*, pages 395–400, Mexico City, Mexico, September 2001.
- [109] W. Lei and W. Qidi. Further example study on ant system algorithm based continuous space optimization. In *Proceedings of the 4th Congress on Intelligent and Automation*, pages 2541–2545, Shangai, P.R. China, 10-14 June 2002.
- [110] E. Marchiori and A. Steenbeck. An Iterated Heuristic Algorithm for the Set Covering Problem. In K. Mehlhorn, editor, *Proceedings WAE '98, Saarbrücken, Germany, August 20-22*, pages 1–3, 1998.
- [111] N. Metropolis, R Bivins, M. Storm, A. Turkevich, J.M. Miller, and G. Friedlander. Monte carlo calculations on intranuclear cascades. In *Physical Review*, volume 110 of *2nd. series*, pages 185–203. 1958.

- [112] Efrén Mezura-Montes and Carlos A. Coello Coello. Conceptos de Optimización Multiobjetivo para el Manejo de Restricciones en Algoritmos Evolutivos: Un Estudio Comparativo. In Salvador Botello, Arturo Hernández, and Carlos A. Coello Coello, editors, *Proceedings of the 1st Mexican Conference on Evolutionary Computation (COMCEV 2003)*, pages 1–12, Guanajuato, México, May 2003. Guanajuato, México, CIMAT, A.C. (In spanish).
- [113] Z. Michalewicz. Genetic algorithms, numerical optimization, and constraints. In L. Eshelman, editor, *Sixth International Conference on Genetic Algorithms*, pages 151–158. Morgan Kauffman, San Mateo, 1995.
- [114] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer Verlag, Berlin, 1999.
- [115] Z. Michalewicz and N. Attia. Evolutionary optimization of constrained problems. In A. V. Sebald and L. J. Fogel, editors, *Third Annual Conference on Evolutionary Programming*, pages 98–108. World Scientific, Singapore, 1994.
- [116] Z. Michalewicz and C. Janikow. Handling constraints in genetic algorithms. In *Proc. of the 4th Int. Conf. on Genetic Algorithms.*, pages 151–157. Morgan Kaufman, 1991.
- [117] Z. Michalewicz, T. Logan, and S. Swaminathan. Evolutionary operators for continuous convex parameter spaces. In A.V. Sebald and L.J. Fogel, editors, *Third Annual Conference on Evolutionary Programming*, pages 84–87, River Edge, NJ, 1994. World Scientific Publishing.
- [118] Zbigniew Michalewicz and David B. Fogel. *How to Solve It: Modern Heuristics*. Springer, Berlin, 2000.

- [119] Zbigniew Michalewicz and G.Nazhiyath. Genocop III: A co-evolutionary algorithm for numerical optimization with nonlinear constraints. In David B. Fogel, editor, *Second IEEE International Conference on Evolutionary Computation*, pages 647–651, Piscataway, New Jersey, 1995. IEEE Press.
- [120] Zbigniew Michalewicz, Girish Nazhiyath, and Maciej Michalewicz. A note on usefulness of geometrical crossover for numerical optimization problems. In Lawrence J. Fogel, Peter J. Angeline, and Thomas Bäck, editors, *Evolutionary Programming V: Proc. of the Fifth Annual Conf. on Evolutionary Programming*, pages 305–311, Cambridge, MA, 1996. MIT Press.
- [121] Zbigniew Michalewicz and Marc Schoenauer. Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation*, 4(1):1–32, 1996.
- [122] N. Monmarché, G. Venturini, and M. Slimane. On how pachycondyla apicalis ants suggest a new search algorithm. *Future Generation Computer Systems*, 16:937–946, 2000.
- [123] H. Myung, J. Kim, and D. Fogel. Preliminary investigation into a two-stage method of evolutionary optimization on constrained problems. In R. G. Reynolds J. R. McDonnell and D. B. Fogel, editors, *Proceedings of the Fourth Annual Conference on Evolutionary Programming*, pages 449–463. MIT Press, 1995.
- [124] I. Osman and J. Kelly. *Methaheuristics: Theory and Practice*. Kluwer Academic Publishers: Norwell MA, 1996.
- [125] Charles C. Palmer and Aaron Kershenbaum. Representing trees in genetic algorithms. In Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz, editors, *Handbook of Evolutionary Computation*, pages G1.3:1–8. Institute of Physics Publishing and Oxford University Press, Bristol, New York, 1997.

- [126] Jerome Pannier and Jin kao Hao. Simulated Annealing and Tabu Search for Constraint Solving. Artificial Intelligence and Mathematics V, Electronic Proceedings (<http://rutcor.rutgers.edu/amai/proceedings.html>), December 15 1998.
- [127] C. H. Papadimitriou and H. R. Lewis. *Elements of the Theory of Computation*. Prentice-Hall, Inc., 1981.
- [128] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization*. Prentice Hall, 1982.
- [129] C. Papadimitriu. *Computational Complexity*. Addison-Wesley, 1994., 1994.
- [130] J. Paredis. Co-evolutionary constraint satisfaction. In *Parallel Problem Solving from Nature*, volume 866 of *Lecture Notes in Computer Science*, pages 46–55, New York, 1994. Springer Verlag.
- [131] I. C. Parmee and G. Purchase. The development of a directed genetic search technique for heavily constrained design spaces. In I. C. Parmee, editor, *Adaptive Computing in Engineering Design and Control- '94*, pages 97–102, Plymouth, UK, 1994. University of Plymouth.
- [132] I.C. Parmee and G. Purchase. The development of a directed genetic search technique for heavily constrained design spaces. In *Adaptive Computing in Engineering Design and Control*, University of Plymouth, September 1994.
- [133] Vrahatis M.N Parsopoulos, K.E. Particle swarm optimization method for constrained optimization problems. In V. Kvasnicka J. Pospichal P. Sincak, J. Vascak, editor, *Intelligent Technologies - Theory and Applications: New Trends in Intelligent Technologies*, volume 76, pages 214–220. 2002.

- [134] D. Powel and M.M. Skolnick. Using genetic algorithms in engineering design optimization with non-linear constraints. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 424–431, San Mateo, CA, 1993. Morgan Kaufman.
- [135] M. Protasi and R. Battiti. Reactive local search techniques for the maximum conjunctive constraint satisfaction problem (k-CCSP). *Discrete Applied Mathematics*, pages 3–27, 1999.
- [136] G. Toscano Pulido and C. Coello-Coello. A constraint-handling mechanism for particle swarm optimization. In *IEEE 2004 Congress on Evolutionary Computation*, pages 1396–1403, June 19-24, Portland, OR. USA, 2004.
- [137] M. Rahoual, R. Hadji, and V. Bachelet. Parallel ant system for the set covering problem. *Lecture Notes in Computer Science*, 2463:262–268, 2002.
- [138] G. R. Raidl. An improved genetic algorithm for the multiconstrained 0-1 knapsack problem. In *Proceedings of the 5th IEEE Conference on Evolutionary Computation at the 1998 IEEE World Congress on Computational Intelligence*, pages 207–211, Anchorage, Alaska, May 1998.
- [139] Tapabrata Ray and K.M. Liew. Society and Civilization: An Optimization Algorithm Based on the Simulation of Social Behavior. *IEEE Transactions on Evolutionary Computation*, 7(4):386–396, August 2003.
- [140] I. Rechenberg. Cybernetic solution path of an experimental problem. Royal Aircraft Establishment, Library translation No. 1122, Farnborough, Hants., UK, August 1965.

- [141] I. Rechenberg. *Evolutionstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, Stuttgart, 1973.
- [142] Colin B. Reeves, editor. *Modern Heuristic Techniques for Combinatorial Problems*. John Wiley & Sons, Great Britain, 1993.
- [143] C.R. Reeves. Genetic algorithms and combinatorial optimization. In J. Rayward-Smith, editor, *Applications of Modern Heuristic Techniques*. Alfred Waller Ltd.
- [144] C. Resende, T. Feo, and G. Mauricio. Greedy Randomized Adaptive Search Problems. *Journal of Global Optimization*, 6:109–133, 1994.
- [145] C. Resende, T. Feo, G. Mauricio, and S. Smith. A Greedy Randomized Adaptive Search Procedure for Maximum Independent Set. *Operations Research*, 42:860–878, 1994.
- [146] R. Reynolds. An introduction to cultural algorithms. In A.V. Sebald and L.J. Fogel, editors, *Proceedings of the 3d Annual Conference on Evolutionary Programming*, pages 131–139, River Edge, New Jersey. World Scientific.
- [147] Robert G. Reynolds, Zbigniew Michalewicz, and Michael J. Cavaretta. Using cultural algorithms for constraint handling in GENOCOP. In *Evolutionary Programming*, pages 289–305, 1995.
- [148] Jon T. Richardson, Mark R. Palmer, Gunar Liepins, and Mike Hilliard. Some Guidelines for Genetic Algorithms with Penalty Functions. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms (ICGA-89)*, pages 191–197, San Mateo, California, June 1989. George Mason University, Morgan Kaufmann Publishers.

- [149] Rodolphe G. Le Riche, Catherine Knopf-Lenoir, and Raphael T. Haftka. A Segregated Genetic Algorithm for Constrained Structural Optimization. In Larry J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA-95)*, pages 558–565, San Mateo, California, July 1995. University of Pittsburgh, Morgan Kaufmann Publishers.
- [150] Rodolphe Le Riche and Frédéric Guyon. Dual Evolutionary Optimization. In Pierre Collet, Cyril Fonlupt, Jin-Kao Hao, Evelyne Lutton, and Marc Schoenauer, editors, *Proceedings of the 5th International Conference on Artificial Evolution (AE 2001)*, pages 281–294, Heidelberg, Germany, October 2001. Le Creusot, France, Springer-Verlag. Lecture Notes in Computer Science Vol. 2310.
- [151] Thomas P. Runarsson and Xin Yao. Stochastic ranking for constrained evolutionary optimization. *IEEE Transactions on Evolutionary Computation*, 4(3):284–294, 2000.
- [152] T.P. Runarsson and Xin Yao. *Evolutionary Optimization*, chapter Constrained Evolutionary Optimization – the penalty function approach, pages 87–113. Kluwer Academic Publishers, Boston, 2002.
- [153] S. Russell and P.Norvin. *Artificial Intelligence: A Modern Approach*. Prentice Hall.
- [154] Stuart J. Russell and Peter Norvig. *Artificial Intelligence. A Modern Approach*. Prentice Hall, Upper Saddle River, New Jersey, 1995.
- [155] D. Schaffer. Multiple objective optimization with vector evaluated genetic algorithms. In *Proceedings of the International Conference on Genetic Algorithms and their Applications*, pages 93–100, 1985.
- [156] J. D. Schaffer. Multiple objective optimization with vector evaluated genetic algorithms. In J. J. Grefenstette, editor, *Proceedings*

- of the First International Conference on Genetic Algorithms and Their Applications*, pages 93–100. Lawrence Erlbaum Associates, Hillsdale, NJ, 1985.
- [157] M. Schoenauer and Z. Michalewicz. Evolutionary computation at the edge of feasibility. In Hans-Michael Voigt, Werner Ebeling, Ingo Rechenberg, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature – PPSN IV*, pages 245–254, Berlin, 1996. Springer.
- [158] Marc Schoenauer and Spyros Xanthakis. Constrained GA optimization. In Stephanie Forrest, editor, *Proc. of the Fifth Int. Conf. on Genetic Algorithms*, pages 573–580, San Mateo, CA, 1993. Morgan Kaufmann.
- [159] Luk Schoofs and Bart Naudts. Ant Colonies are Good at Solving Constraint Satisfaction Problems. In *Proceedings of the Congress on Evolutionary Computation 2000 (CEC'2000)*, volume 2, pages 1190–1195, Piscataway, New Jersey, July 2000. IEEE Service Center.
- [160] H.-P. Schwefel. *Kybernetische Evolution als Strategie der experimentellen Forschung in der Strömungstechnik*. Diplomarbeit, Technische Universität Berlin, 1965.
- [161] H.-P. Schwefel. *Numerical Optimization of Computer Models*. Wiley, Chichester, 1981.
- [162] R.M. Silva and G.L Ramalho. Ant system for the set covering problem. In *IEEE International Conference on Systems, Man, and Cybernetics*, volume 5, pages 3129–3133, 2001.
- [163] Alice E. Smith and David M. Tate. Genetic optimization using A penalty function. In Stephanie Forrest, editor, *Proceedings of the*

- 5th International Conference on Genetic Algorithms*, pages 499–505, San Mateo, CA, USA, July 1993. Morgan Kaufmann.
- [164] T. Sudkamp. *Languages and Machines: An Introduction to the Theory of Computer Science*. Addison-Wesley, Reading, Massachusetts, 1997.
- [165] Patrick D. Surry and Nicholas J. Radcliffe. The COMOGA Method: Constrained Optimisation by Multiobjective Genetic Algorithms. *Control and Cybernetics*, 26(3):391–412, 1997.
- [166] Patrick D. Surry, Nicholas J. Radcliffe, and Ian D. Boyd. A Multi-Objective Approach to Constrained Optimisation of Gas Supply Networks : The COMOGA Method. In Terence C. Fogarty, editor, *Evolutionary Computing. AISB Workshop. Selected Papers*, pages 166–180, Sheffield, U.K., 1995. Springer-Verlag.
- [167] Eric Taillard, Philippe Badeau, Michel Gendreau, François Guertin, and Jean-Yves Potvin. A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation science*, 31:170–186, 1997.
- [168] K.C. Tan, T.H. Lee, E.F. Khor, C.M. Heng, and D. Khoo. Non-linear Constraint Handling Techniques via Angular Transformation. In Lee Spector, Erik D. Goodman, Annie Wu, W.B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2001)*, pages 655–662, San Francisco, California, July 2001. Morgan Kaufmann Publishers.
- [169] K. Schittkowski W. Hock. Test examples for nonlinear programming codes. Lecture Notes in Econ. and Math. and Syst. Springer-Verlag, Berlin, Germany, 1981.

- [170] D. Waagen, P. Diercks, and J. McDonnell. The stochastic direction set algorithm: A hybrid technique for finding function extrema. In *First Annual Conference on Evolutionary Programming*, pages 35–42.
- [171] Benjamin W. Wah and Yixin Chen. Hybrid constrained simulated annealing and genetic algorithms for nonlinear constrained optimization. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, pages 925–932, COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea, 27-30 2001. IEEE Press.
- [172] B.W. Wah and Y. Chen. Constrained genetic algorithms and their applications in nonlinear constrained optimization. In *Proceedings International Conference on Tools with Artificial Intelligence*, pages 286–293, November 2000.
- [173] B.W. Wah and T. Wang. Constrained simulated annealing with applications in nonlinear continuous constrained global optimization. In *IEEE International Conference on Tools with Artificial Intelligence*, pages 331–338, November.
- [174] D. Whitley. The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 116–121. Morgan Kaufmann Publishers, San Mateo, CA, 1989.
- [175] D. Whitley and J. Kauth. GENITOR: A different genetic algorithm. Technical Report CS-88-101, Department of Computer Science, Colorado State University, Fort Collins, CO, March 1988.
- [176] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

- [177] Z.Y. Wu and A.R. Simpson. A self-adaptive boundary search genetic algorithm and its application to water distribution systems. *Journal of Hydraulic Research*, 40(2):191–203, 2002.
- [178] M. Yuchi and J-H. Kim. Grouping-based evolutionary algorithms: Seeking balance between feasible and infeasible individuals of constrained optimization problems. In *IEEE 2004 Congress on Evolutionary Computation*, pages 280–287, June 19-24, Portland, OR. USA, 2004.