UNIVERSIDAD
DE MÁLAGA

LENGUAJES Y
CIENCIAS DE LA
COMPUTACIÓN
UNIVERSIDAD DE MÁLAGA

TESIS DOCTORAL

Diseño e Implementación de Algoritmos Genéticos Celulares para Problemas Complejos

Autor
Bernabé Dorronsoro Díaz

Director
Dr. Enrique Alba Torres

Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA

Diciembre de 2006

El Dr. **Enrique Alba Torres**, Titular de Universidad del Departamento de Lenguajes y Ciencias de la Computación de la Universidad de Málaga,

Certifica

que D. **Bernabé Dorronsoro Díaz**, Ingeniero en Informática por la Universidad de Málaga, ha realizado en el Departamento de Lenguajes y Ciencias de la Computación de la Universidad de Málaga, bajo su dirección, el trabajo de investigación correspondiente a su Tesis Doctoral titulada

Diseño e Implementación de Algoritmos Genéticos Celulares para Problemas Complejos

Revisado el presente trabajo, estimo que puede ser presentado al tribunal que ha de juzgarlo, y autorizo la presentación de esta Tesis Doctoral en la Universidad de Málaga.

En Málaga, Diciembre de 2006

Firmado: Dr. Enrique Alba Torres
Titular de Universidad
Dpto. de Lenguajes y Ciencias de la Computación
Universidad de Málaga

Agradecimientos

Son muchas las personas que han contribuido de alguna manera en este trabajo, y a las que estoy muy agradecido. En primer lugar, quiero agradecerle a mi director el Dr. Enrique Alba la dedicación, la paciencia y la ayuda que me ha prestado en todo momento durante estos años. También a mis compañeros de laboratorio, que siempre se han ofrecido cuando los he necesitado. No sería justo olvidar al Dr. Antonio J. Nebro, que al igual que mi director y mis compañeros de laboratorio, ha contribuido en gran medida en este trabajo.

Un agradecimiento muy especial a Patricia, por su incondicional apoyo. Porque me ha ayudado mucho en este tiempo, y me ha comprendido todos esos fines de semana que he tenido que trabajar.

Para terminar, aunque no menos importante, me gustaría dedicar aquí un pequeño espacio para mi familia. A mis padres, por su insistente respaldo y su confianza en mi. Sin ellos no hubiera podido realizar este trabajo. También quiero mencionar a mis hermanos y sus familias que siempre me han apoyado mucho.

Índice de Contenidos

1. Introducción **1**
1.1. Planteamiento . . . 1
1.2. Objetivos y Fases . . . 2
1.3. Aportaciones . . . 4
1.4. Organización de la Tesis . . . 5

I Modelo Algorítmico **7**

2. Introducción a los Algoritmos Evolutivos **9**
2.1. Optimización y Algoritmos Avanzados . . . 10
2.2. Resolución de Problemas con Metaheurísticas . . . 12
2.3. Algoritmos Evolutivos . . . 12
2.4. Algoritmos Evolutivos Descentralizados . . . 15
2.5. Algoritmos Evolutivos Celulares . . . 17
2.5.1. Algoritmos Evolutivos Celulares Síncronos y Asíncronos . . . 20
2.5.2. Caracterización Formal de la Población en cEAs . . . 21
2.6. Algoritmos Genéticos Celulares . . . 22
2.7. Críticas al Estado del Arte de los cEAs . . . 24
2.8. Conclusiones . . . 25

3. El Estado del Arte en Algoritmos Evolutivos Celulares **27**
3.1. EAs Celulares: un Nuevo Modelo Algorítmico . . . 27
3.2. La Investigación en la Teoría de los Modelos Celulares . . . 28
3.2.1. Caracterizando el Comportamiento de cEAs . . . 30
3.2.2. La Influencia de la Ratio . . . 31
3.3. Estudios Empíricos sobre el Comportamiento de cEAs . . . 32
3.4. Mejoras Algorítmicas al Modelo Canónico . . . 33
3.5. Modelos Paralelos de cEAs . . . 36
3.6. Conclusiones . . . 38

4. Diseño Algorítmico y Experimental **39**
4.1. Formalización de GA Descentralizado . . . 39
4.2. Propuesta de Nuevos Modelos Eficientes . . . 42

4.3. Evaluación de los Resultados . . . 45
4.3.1. El Caso Mono-Objetivo . . . 45
4.3.2. El Caso Multi-Objetivo . . . 47
4.3.3. Algunas Definiciones Adicionales . . . 49
4.4. Conclusiones . . . 50

5. Nueva Propuesta Teórica: Estudios sobre la Presión de Selección en cGAs 51
5.1. La Presión de Selección . . . 52
5.2. Estudio Teórico . . . 54
5.2.1. Aproximación al Modelo Determinista . . . 54
5.2.2. Modelos Existentes de Aproximación al Comportamiento Real de cGAs . . . 56
5.2.3. Dos Nuevos Modelos de Aproximación . . . 60
5.2.4. Comparación de los Mejores Modelos . . . 65
5.3. Estudio Práctico . . . 67
5.3.1. Validación sobre Optimización Combinatoria . . . 68
5.3.2. Validación sobre Optimización Continua . . . 72
5.4. Conclusiones . . . 74

II Metodología 77

6. Diseño de cGAs Adaptativos 79
6.1. Introducción . . . 79
6.2. Algunas Nociones de Auto-Adaptación . . . 80
6.3. Descripción de los Algoritmos Utilizados . . . 81
6.3.1. Algoritmos Estáticos y Pre-Programados . . . 83
6.3.2. Algoritmos Adaptativos . . . 84
6.4. Experimentación . . . 87
6.4.1. Parametrización . . . 88
6.4.2. Estudiando los Efectos de Estructurar la Población . . . 89
6.4.3. Resultados Experimentales Utilizando Torneo Binario . . . 89
6.4.4. Resultados Experimentales Utilizando Ruleta . . . 92
6.4.5. Discusión Adicional . . . 95
6.5. Modelos Avanzados de cGAs . . . 99
6.6. Conclusiones . . . 103

7. Diseño de Algoritmos Genéticos Jerárquicos Celulares 105
7.1. Introducción . . . 105
7.2. El Algoritmo HcGA . . . 106
7.2.1. Jerarquía . . . 107
7.2.2. Selección por Disimilitud . . . 108
7.3. Primeros Resultados Teóricos: Tiempo de Takeover . . . 108
7.3.1. Ajuste de las Curvas de Takeover . . . 110
7.4. Experimentos Computacionales . . . 111
7.5. Conclusiones . . . 115

8. Diseño de Algoritmos de Estimación de Distribución Celulares 117
8.1. Algoritmos de Estimación de Distribución . . . 118
8.2. Algoritmos de Estimación de Distribución Celulares . . . 120
8.3. Experimentación . . . 121
8.3.1. Algoritmo de Distribución Marginal de una sola Variable . . . 121
8.3.2. Resultados . . . 123
8.4. Conclusiones . . . 125

9. Diseño de Algoritmos Meméticos Celulares 127
9.1. Algoritmos Meméticos Celulares . . . 128
9.2. Componentes Canónicos y Avanzados en cMAs . . . 128
9.2.1. Tres Técnicas de Búsqueda Local Básica para SAT . . . 128
9.2.2. Algoritmos Meméticos Celulares Para SAT . . . 131
9.3. Análisis Computacional . . . 132
9.3.1. Los Efectos de Estructurar la Población y Utilizar una Función de Adecuación con Pesos Adaptativos (SAW) . . . 132
9.3.2. Resultados: Heurísticas no Meméticas para SAT . . . 134
9.3.3. Resultados: cMAs . . . 135
9.3.4. Comparación con otros Resultados en la Literatura . . . 137
9.4. Conclusiones . . . 138

10. Diseño de Algoritmos Genéticos Celulares Paralelos 139
10.1. Algoritmo Genético Meta-Celular . . . 140
10.1.1. Parametrización . . . 140
10.1.2. Presentación y Análisis de los Resultados . . . 141
10.2. Algoritmo Genético Distribuido Celular . . . 142
10.2.1. Parametrización . . . 143
10.2.2. Presentación y Análisis de los Resultados . . . 146
10.3. Conclusiones . . . 149

11. Diseño de Algoritmos Genéticos Celulares para Optimización Multi-objetivo 151
11.1. Fundamentos de la Optimización Multi-objetivo . . . 152
11.2. El Algoritmo MOCell . . . 153
11.3. Resultados Computacionales . . . 154
11.3.1. Métricas de Rendimiento . . . 156
11.3.2. Discusión de los Resultados . . . 156
11.4. Nuevos Aspectos de Diseño en MOCell . . . 158
11.5. Conclusiones . . . 163

12. Implementación de cGAs: La Biblioteca JCell 165
12.1. La Biblioteca JCell . . . 165
12.2. Utilización de JCell . . . 170
12.3. Conclusiones . . . 173

III Aplicaciones de los Algoritmos Genéticos Celulares 175

13. Caso Discreto: El Problema de Rutas de Vehículos 177
13.1. Problema de Rutas de Vehículos . . . 179
13.2. El Algoritmo Propuesto . . . 180
13.2.1. Representación del Problema . . . 181
13.2.2. Recombinación . . . 182
13.2.3. Mutación . . . 183
13.2.4. Búsqueda Local . . . 184
13.3. Buscando un Nuevo Algoritmo: El Camino hacia JCell2o1i . . . 185
13.3.1. Algoritmo Genético Celular vs. Generacional . . . 186
13.3.2. La Importancia del Operador de Mutación . . . 187
13.3.3. Ajustando la Etapa de Búsqueda Local . . . 189
13.4. Resolviendo CVRP con JCell2o1i . . . 192
13.4.1. Problemas de Augerat et al. . . . 193
13.4.2. Problemas de Van Breedam . . . 194
13.4.3. Problemas de Christofides y Eilon . . . 196
13.4.4. Problemas de Christofides, Mingozzi y Toth . . . 197
13.4.5. Problemas de Fisher . . . 198
13.4.6. Problemas de Golden, Wasil, Kelly y Chao . . . 198
13.4.7. Problemas de Taillard . . . 199
13.4.8. Problemas de Instancias Tomadas del TSP . . . 200
13.5. Nuevas Soluciones al Problema VRP . . . 201
13.6. Conclusiones . . . 202

14. Caso Continuo: Conjunto Clásico de Problemas de Optimización Continua 203
14.1. Experimentación . . . 203
14.1.1. Ajustando el Algoritmo . . . 205
14.1.2. Comparación con Otros Algoritmos . . . 206
14.2. Conclusiones . . . 209

15. Caso Real: Optimización del Proceso de Difusión en MANETs 211
15.1. Introducción . . . 211
15.2. El Problema . . . 212
15.2.1. Redes Móviles Ad Hoc Metropolitanas. El Simulador Madhoc . . . 212
15.2.2. Inundación con Retrasos y Vecindarios Acumulativos . . . 214
15.2.3. Definición de MOPs . . . 216
15.3. Un cGA Multi-Objetivo: cMOGA . . . 217
15.3.1. Tratando con Restricciones . . . 218
15.4. Experimentos . . . 218
15.4.1. Parametrización de cMOGA . . . 219
15.4.2. Configuración de Madhoc . . . 219
15.4.3. Resultados para DFCNT . . . 221
15.4.4. Resultados para cDFCNT . . . 224
15.4.5. Comparando DFCNT y cDFCNT . . . 226
15.5. Comparación de cMOGA con Otras Metaheurísticas . . . 226

15.5.1. Parametrización . . . 226
15.5.2. Discusión . . . 227
15.6. Conclusiones . . . 230

IV Conclusiones y Trabajos Futuros 231

V Apéndices 241

A. Otros Problemas Estudiados 243
A.1. Problemas de Optimización Combinatoria . . . 243
A.1.1. Problema COUNTSAT . . . 243
A.1.2. Problema del Diseño de Códigos Correctores de Errores – ECC . . . 244
A.1.3. Problema de la Modulación en Frecuencia de Sonidos – FMS . . . 245
A.1.4. Problema IsoPeak . . . 245
A.1.5. Problema del Máximo Corte de un Grafo – MAXCUT . . . 245
A.1.6. Problema Masivamente Multimodal – MMDP . . . 246
A.1.7. Problema de las Tareas de Mínima Espera – MTTP . . . 247
A.1.8. Problema OneMax . . . 247
A.1.9. Problema Plateau . . . 248
A.1.10. Problema P-PEAKS . . . 248
A.1.11. Problema de Satisfacibilidad – SAT . . . 248
A.2. Problemas de Optimización Continua . . . 249
A.2.1. Problemas Académicos . . . 249
A.2.2. Problemas del Mundo Real . . . 251
A.3. Problemas de Optimización Multi-objetivo . . . 252
A.4. Conclusiones . . . 254

B. Resultados y Mejores Soluciones Encontradas para VRP 255
B.1. Mejores Soluciones Encontradas . . . 255
B.2. Detalles Numéricos Exhaustivos para VRP . . . 263

C. Relación de Publicaciones que Sustentan esta Tesis Doctoral 267

Índice de Términos 294

Capítulo 1

Introducción

1.1. Planteamiento

El diseño de algoritmos cada vez más eficientes para resolver problemas complejos (tanto problemas de optimización como de búsqueda) ha sido tradicionalmente uno de los aspectos más importantes en la investigación en el campo de la informática. El objetivo perseguido en este campo es fundamentalmente el desarrollo de nuevos métodos capaces de resolver los mencionados problemas complejos con el menor esfuerzo computacional posible, mejorando así a los algoritmos existentes. En consecuencia, esto no sólo permite afrontar los problemas considerados de forma más eficiente, sino afrontar tareas vedadas en el pasado debido a su alto coste computacional.

En este contexto, la actividad investigadora tanto en algoritmos exactos como heurísticos para resolver problemas complejos de optimización está creciendo de forma evidente en estos días. La razón es que continuamente se están afrontando nuevos problemas de ingeniería, mientras que al mismo tiempo cada vez están disponibles mejores recursos computacionales, como nuevos tipos de ordenadores, redes, y entornos como Internet.

La aplicación de Algoritmos Evolutivos (EAs) a problemas de optimización ha sido muy intensa durante la última década [40]. Un EA es un proceso iterativo que trabaja sobre un conjunto P de individuos que componen una población, y a los que se les aplica una serie de operadores de variación que permiten a la población de individuos evolucionar y mejorar. Los individuos representan soluciones potenciales al problema considerado, a las que se les asocia un valor de adecuación, también llamado de *fitness*, con el fin de medir cómo de buena es la solución representada por el individuo para el problema dado. Estos algoritmos son usualmente empleados para resolver problemas complejos tales como tareas de optimización con restricciones, con ruido, engañosos, o con un elevado grado de epistasis y/o multimodalidad [15].

El comportamiento que muestra un EA al resolver los problemas viene determinado por el equilibrio que mantiene al manipular las soluciones de la población entre la explotación local de las mejores soluciones y la exploración del campo de búsqueda. Si un EA tiene una elevada capacidad de explotación, los individuos (soluciones potenciales) evolucionarán muy rápidamente hacia soluciones mejores cercanas en el espacio de búsqueda que, muy probablemente, no serán soluciones óptimas al problema, y de las que le resultará muy difícil escapar. La consecuencia de esta rápida progresión de las soluciones es una fuerte pérdida de diversidad en la población (todas las soluciones que la componen se vuelven muy parecidas, o incluso similares), por lo que la evolución de los individuos se estanca (es decir, los operadores de variación no pueden generar individuos mejores que los existentes). Por el contrario, un algoritmo que potencie demasiado la exploración recorrerá una parte considerablemente amplia del espacio de búsqueda, pero no será capaz de profundizar en las zonas más prometedoras, por lo que no encontrará soluciones de calidad.

Tradicionalmente, la mayoría de los EAs trabajan sobre una única población (EAs *panmícticos*), aunque existe una reciente tendencia que consiste en estructurar la población mediante la definición de algún criterio de vecindad entre las soluciones intermedias manejadas por el algoritmo [27]. Los EAs estructurados presentan de forma natural distintas regiones del campo de búsqueda en la población, permitiendo de esta forma realizar una búsqueda descentralizada. El efecto perseguido de estructurar la población consiste en el mantenimiento de la diversidad de soluciones durante más tiempo, mejorando de esta manera la capacidad de exploración del algoritmo en el espacio de búsqueda, mientras que la explotación puede también ser reforzada a veces muy fácilmente. Por tanto, estos algoritmos se comportan mejor en muchos casos que sus algoritmos equivalentes con poblaciones en panmixia, mejorando el comportamiento numérico y su potencial [27]. Entre los EAs estructurados, los más comúnmente conocidos son los distribuidos y los celulares. En los EAs distribuidos (dEAs), la población se encuentra particionada en algunas sub-poblaciones de menor tamaño que evolucionan independientemente unas de otras, y que intercambian información con una frecuencia determinada. En el caso de los EAs celulares (cEAs) la población está estructurada utilizando el concepto de vecindario [27], de forma que los individuos sólo pueden interactuar con sus vecinos más próximos en la población.

La presente tesis doctoral se encuentra dentro de este marco. En concreto, todo el trabajo desarrollado en esta tesis trata sobre cEAs. Los cEAs surgieron como una evolución de un modelo de *autómata celular* (CA) aplicado sobre un EA [272, 288]. Los CAs son sistemas dinámicos formados por un conjunto de celdas que pueden estar en varios estados predefinidos. Estas celdas evolucionan en función de los estados de las celdas vecinas según un conjunto de reglas establecidas. Los CAs han sido ampliamente estudiados en la literatura, por lo que la posibilidad de importar las ideas existentes en el campo de los CAs a los cEAs nos motiva especialmente. Además, otro aspecto importante es que los cEAs están siendo aplicados con mucho éxito en los últimos años en la resolución de problemas complejos académicos [9, 103, 172] e industriales, como problemas de logística [17] o de ingeniería [11].

Teniendo esto en mente, proponemos en este trabajo en primer lugar caracterizar el comportamiento de los cEAs de forma tanto teórica como práctica. Consideramos que ésta es una parte importante en nuestro trabajo, puesto que en la actualidad sólo existen estudios breves y dispersos en relación con este tema. En segundo lugar, nos planteamos proponer extensiones eficientes y precisas al modelo canónico de cEA. En concreto, nuestro trabajo se basa en explorar nuevas propuestas de cEAs puros, híbridos, paralelos, distribuidos, con población jerárquica, nuevas familias de EAs celulares (en concreto EDAs celulares), o incluso multi-objetivo (en los que se considera la optimización de varias funciones usualmente en conflicto) para mejorar los resultados existentes en problemas pertenecientes a los dominios académico e industrial. Todos los nuevos cEAs propuestos en este trabajo son comparados con cEAs canónicos y con algoritmos pertenecientes al estado del arte para una plétora de problemas complejos de optimización que pertenecen a los campos de optimización combinatoria, de programación con números enteros, continua, o multi-objetivo.

1.2. Objetivos y Fases

En esta tesis, el objetivo fundamental es crear el cuerpo de conocimiento de los algoritmos evolutivos celulares y avanzar en el estado del arte de dicho campo. Para ello, seguiremos las fases típicas que establece la filosofía del método científico [112, 286], según definió F. Bacon:

1. **Observación:** Observar es aplicar atentamente los sentidos a un objeto o a un fenómeno, para estudiarlos tal como se presentan en realidad. En nuestro caso, esta fase consistirá en el estudio del estado-del-arte en cEAs.

2. **Inducción:** La acción y efecto de extraer, a partir de determinadas observaciones o experiencias, el principio particular de cada una de ellas. En este paso, estudiaremos los trabajos presentados en el punto anterior, criticándolos, y poniendo de manifiesto sus principales carencias.

3. **Hipótesis:** Planteamiento mediante la observación siguiendo las normas establecidas por el método científico. En nuestro caso, proponemos ideas que pueden resolver los problemas anteriormente manifestados.

4. **Probar la hipótesis por experimentación:** Implementaremos nuevos modelos algorítmicos basándonos en las mejoras propuestas en nuestra hipótesis de trabajo, y realizaremos nuestros experimentos.

5. **Demostración o refutación de la hipótesis:** Analizaremos los resultados obtenidos con los nuevos modelos desarrollados, comparándolos exhaustivamente con los resultados de los algoritmos canónicos existentes. Además, en los casos que ha sido posible también se han comparado nuestros nuevos modelos con algunos otros algoritmos pertenecientes al estado del arte del dominio dado.

6. **Conclusiones:** Presentaremos las conclusiones a las que llegamos tras nuestro trabajo de investigación, y sugeriremos algunas líneas de trabajo futuro que surgen de nuestro estudio.

El método científico se sustenta en dos pilares fundamentales: la reproducibilidad y la falsabilidad, que establece que toda proposición científica tiene que ser susceptible de ser falsada.

En relación a la reproducibilidad, presentamos en todo momento todos los detalles necesarios para que puedan ser reproducidos los experimentos aquí presentados por cualquier otro investigador que tenga interés en alguna de nuestras aportaciones. Además, facilitamos de forma gratuita en Internet una biblioteca de programación que ha sido desarrollada durante este trabajo de tesis, y con la que se han realizado la gran mayoría de los experimentos presentados.

En cuanto a la falsabilidad, ofrecemos en todos nuestros estudios los resultados obtenidos de forma clara, estructurada y sencilla. Debido a la naturaleza estocástica de los algoritmos sobre los que trabajamos, se han realizado un mínimo de 30 (usualmente 100) experimentos independientes, y los resultados son presentados como la media de los resultados obtenidos en todos los experimentos. Además, para asegurar la relevancia estadística de nuestras conclusiones, aplicamos un conjunto de análisis estadísticos a nuestros datos en todos los estudios realizados.

Con el fin de llevar a cabo nuestro objetivo, nos planteamos la realización del siguiente conjunto de sub-objetivos:

- Descripción unificada de los algoritmos evolutivos celulares (cEAs).
- Estudio y clasificación de las propuestas de cEAs existentes en la literatura. Propuesta de una nueva taxonomía de cEA.
- Análisis teórico y práctico para comprender este tipo de algoritmos.
- Análisis y crítica de los modelos existentes.
- Propuesta de nuevos modelos de cEAs que sean a la vez eficientes y precisos.
- Aplicación de las mejoras algorítmicas y los nuevos modelos propuestos a problemas complejos y reales, pertenecientes a los campos de la optimización combinatoria, continua o multi-objetivo, en dominios de interés real como las telecomunicaciones, la logística, o el diseño.

1.3. Aportaciones

En este trabajo de tesis se aportan resultados a la literatura de cEAs tanto en el campo teórico como en el práctico. Para ello, la contribución científica que pretendemos hacer en investigación con nuestro estudio comprende básicamente los siguientes puntos:

1. Análisis y clasificación de los modelos de algoritmos evolutivos celulares existentes en la literatura.
2. Caracterización de los cEAs, presentando un modelo matemático que, a diferencia de los existentes hasta ahora, caracterice los cEAs en función del valor de la ratio entre los radios (medida de la forma y tamaño) del vecindario y la población del algoritmo. Además, se ha realizado una validación de este estudio en la práctica con el fin de corroborar los resultados creados en el campo teórico.
3. Diseño e implementación, siguiendo el paradigma orientado a objetos, de una biblioteca (disponible públicamente) que implementa tres modelos de algoritmo genético (celular, estado estacionario y generacional), así como la mayoría de las nuevas propuestas y mejoras algorítmicas presentadas en este trabajo de tesis.
4. Diseño de un nuevo modelo de algoritmo evolutivo celular con ratio dinámica adaptativa, de forma que adapte automáticamente durante la ejecución el equilibrio entre exploración y explotación en función de la evolución de la diversidad en la población a lo largo de las sucesivas generaciones.
5. Implementación por primera vez de una nueva clase de cEAs en los que se establece una jerarquía entre los individuos que componen la población.
6. Implementación del primer algoritmo de estimación de distribuciones celular, nunca antes propuesto en tesis doctoral alguna.
7. Implementación del primer algoritmo memético celular.
8. Implementación de nuevos modelos de cEAs paralelos y distribuidos en un sistema de *grid computing* [4, 43, 107].
9. Propuesta, diseño y estudio del primer modelo ortodoxo de algoritmo evolutivo celular aplicado al campo de la optimización multi-objetivo.
10. Aplicaciones de los algoritmos evolutivos celulares a problemas muy complejos de optimización combinatoria y continua pertenecientes a campos como las telecomunicaciones, la logística o el diseño. Validación de su potencia real y definición de sus ventajas.
11. Definición de nuevos problemas en el campo de las telecomunicaciones y la optimización multi-objetivo.
12. Encontrar la nueva mejor solución para diez instancias del problema de rutas de vehículos (VRP).

Como ya se ha mencionado, durante la elaboración de esta tesis doctoral se ha desarrollado una biblioteca de programación de forma que todos los resultados publicados en este documento se puedan reproducir fácilmente con un esfuerzo realmente bajo. Esta biblioteca se llama JCell, y permite al usuario trabajar con algoritmos genéticos celulares (cGAs), un tipo de cEAs, en campos innovadores de investigación. Únicamente el caso de los algoritmos genéticos está implementado en JCell, pero puede ser

extendida fácilmente a cualquier otro tipo de algoritmo evolutivo. El uso de JCell es muy sencillo, ya que para ello sólo es necesaria la manipulación de un simple fichero de configuración. Además, consideramos que JCell es una herramienta realmente interesante y útil para investigaciones futuras por parte de otros científicos, ya que permite la combinación de todas las prometedoras técnicas desarrolladas en esta tesis.

Finalmente, se ha llevado a cabo una importante labor de diseminación del contenido de las investigaciones desarrolladas en este trabajo. Para tal fin, además de las publicaciones aparecidas en congresos, revistas y libros tanto nacionales como internacionales, se han desarrollado páginas web de dominio público para permitir a cualquiera que esté interesado acceder a los resultados de nuestras investigaciones, junto a las de los principales investigadores. En concreto, se ha creado un sitio web acerca del problema VRP [84] y otro que se centra en los cEAs en general [85]. Ambos sitios web gozan de un considerable reconocimiento, superando entre las dos los 150 visitantes diarios, en media; tienen enlaces desde multitud de páginas especializadas, y han sido ya referenciados en numerosos trabajos.

1.4. Organización de la Tesis

Este documento de tesis se estructura en cuatro partes fundamentales. En la primera se presenta una introducción general al campo de estudio de esta tesis, un extenso estado del arte de las principales aportaciones que existen en la literatura de dicho campo, y un crítica a dicho estado del arte, poniendo de manifiesto los problemas con los que cuentan las técnicas actuales. La segunda parte de la tesis está enfocada en la aportación de las nuevas técnicas propuestas, que han sido desarrolladas con el fin de solventar parte de los aspectos criticados en la parte primera. La tercera parte de este documento se centra en el estudio experimental de algunas aplicaciones de las técnicas desarrolladas a problemas muy complejos, muchos de ellos tomados del mundo real. Finalmente, en la cuarta parte mostramos las principales conclusiones que se desprenden de este trabajo, así como las líneas de trabajo futuro más inmediatas que surgen de nuestro estudio.

- **PARTE I. Modelo Algorítmico.** En el Capítulo 2 se presenta una introducción general al campo de las metaheurísticas, centrándose en los algoritmos evolutivos celulares, un tipo de algoritmo evolutivo que se caracteriza principalmente por su población estructurada. Dentro de los algoritmos evolutivos celulares, prestaremos especial atención a los algoritmos genéticos celulares, que son el objeto de estudio en esta tesis. Al final del capítulo, se presenta una crítica de los principales problemas existentes en el campo de los algoritmos genéticos celulares.

  El Capítulo 3 está centrado en el estudio del estado del arte de los algoritmos evolutivos celulares, presentando un amplio resumen de las principales publicaciones que han aparecido en la literatura estudiando este tipo de algoritmos.

  Proponemos en el Capítulo 4 el diseño de los algoritmos que se desarrollarán en los siguientes capítulos como nuestra aportación para resolver los problemas existentes en el estado del arte. Así mismo, se presenta en este capítulo el diseño de nuestros experimentos, en el que mostraremos la manera en que se procederá para la evaluación y comparación de los algoritmos.

  Para terminar esta primera parte, el Capítulo 5 contiene un estudio teórico del comportamiento de los algoritmos evolutivos celulares, así como un estudio empírico que sustentará nuestras conclusiones. Consideramos que este capítulo es interesante, puesto que existen muy pocos trabajos en los que se trate de caracterizar el comportamiento de los cGAs de forma teórica, y en ninguno de ellos se llega a profundizar tanto como se ha hecho en este capítulo.

- **PARTE II. Metodología.** En el Capítulo 6 se propone una nueva familia de algoritmos evolutivos celulares, que se caracteriza por adaptar de forma automática las capacidades de búsqueda del algoritmo durante la ejecución. El Capítulo 7 presenta un nuevo tipo de algoritmos genéticos celulares en los que se establece una jerarquía en la población, de forma que se pueda mantener la diversidad por más tiempo con respecto a un algoritmo genético celular tradicional, a la vez que se promociona la explotación de las zonas más prometedoras del espacio de búsqueda.

  Se presentan en el Capítulo 8 los algoritmos de estimación de distribuciones celulares, un nuevo tipo de algoritmo evolutivo celular desarrollado durante los trabajos de esta tesis. El Capítulo 9 describe los algoritmos meméticos celulares, otro nuevo tipo de algoritmo genético celular que presentamos en este trabajo de tesis, y que se caracteriza por incorporar técnicas que nos permiten utilizar conocimiento del problema para así construir un algoritmo altamente especializado para el problema en cuestión. Esto nos permitirá idealmente mejorar los resultados obtenidos para un problema dado con respecto a un algoritmo genético celular tradicional.

  El Capítulo 10 está dedicado al estudio de nuevas técnicas de paralelización de algoritmos evolutivos celulares, proponiéndose dos técnicas concretas (en entornos de área de red local y de redes distribuidas), que serán evaluadas. Finalmente, se presenta en el Capítulo 11 una nueva aportación al campo de los algoritmos genéticos celulares, consistente en la adaptación de este tipo de algoritmos al campo de la optimización multi-objetivo.

  Por último, en el Capítulo 12 se presenta la descripción de la biblioteca de algoritmos genéticos celulares que se ha desarrollado durante el trabajo en esta tesis doctoral.

- **PARTE III. Aplicaciones de los Algoritmos Genéticos Celulares.** En esta tercera parte se presentan varias aplicaciones de los algoritmos genéticos celulares. En concreto, el Capítulo 13 está dedicado al estudio del comportamiento de un algoritmo memético celular sobre un problema de optimización combinatoria (problema discreto) con aplicaciones directas al campo de la logística. En el Capítulo 14 se analiza el rendimiento de los algoritmos genéticos celulares para una batería amplia de problemas de optimización numérica. Por último, el Capítulo 15 está dedicado a la resolución de un problema multi-objetivo perteneciente al campo de las telecomunicaciones (problema con variables discretas y continuas) con un algoritmo genético celular.

- **PARTE IV. Conclusiones y Trabajos Futuros.** En esta última parte se presentan las principales conclusiones que se desprenden de los estudios realizados para este trabajo de tesis. Así mismo, se describen también las principales líneas de trabajo futuro que surgen del presente estudio.

Parte I

Modelo Algorítmico

Capítulo 2

Introducción a los Algoritmos Evolutivos

Como ya se ha comentado con anterioridad, la investigación en algoritmos exactos, heurísticos y metaheurísticas para resolver problemas de optimización combinatoria tiene una vigencia inusualmente importante en estos días. La principal ventaja de la utilización de algoritmos exactos es que garantizan encontrar el óptimo global de cualquier problema, pero tienen el grave inconveniente de que en problemas reales (NP-Completos) su tiempo de ejecución crece de forma exponencial con el tamaño de la instancia. En cambio los heurísticos suelen ser bastante rápidos, pero cuentan con el hándicap de la mediocridad que, generalmente, tienen las soluciones obtenidas. Además son algoritmos muy complejos de definir para muchos problemas. En cambio, las metaheurísticas ofrecen un equilibrio entre ambos extremos; son métodos genéricos que ofrecen una buena solución (incluso en muchos casos el óptimo global) en un tiempo de cómputo moderado.

El gran desarrollo que se está dando en el mundo de la informática en los últimos años está permitiendo que puedan afrontarse problemas cada vez más complejos. En la literatura existe un gran número de metaheurísticas diseñadas para resolver este tipo de problemas complejos. De entre estas metaheurísticas, los algoritmos evolutivos (EAs) son unas de las técnicas mejores y más conocidas. Los Algoritmos Evolutivos (EA) están inspirados en la selección natural y la evolución que se dan en la naturaleza, de forma que las especies que forman un ecosistema son capaces de adaptarse a los cambios de su entorno. Esta familia de técnicas sigue un proceso iterativo y estocástico que opera sobre un conjunto de individuos (población), donde cada individuo representa una solución potencial al problema que se está resolviendo. A los individuos se les asigna un valor de adecuación (o fitness) como una medida de su aptitud para el problema en consideración. Este valor representa la información cuantitativa que el algoritmo usa para guiar la búsqueda. La relación entre la explotación de buenas soluciones (potenciada en la fase de selección) y la exploración de nuevas zonas del espacio de búsqueda (que se generan en la fase de evolución) que aplican los EAs sobre el espacio de búsqueda es uno de los factores clave de su alto rendimiento con respecto a otras metaheurísticas. Además, esta relación entre exploración y explotación se puede mejorar de forma importante estructurando la población, lo que nos permite realizar una búsqueda descentralizada. También es posible afinar esta relación entre exploración y explotación con otros parámetros del algoritmo, como por ejemplo los operadores de variación utilizados, o la probabilidad de aplicarlos.

Un tipo de EA con población descentralizada es el de los EAs celulares (cEAs), principal objeto de estudio de esta tesis. Los cEAs son un buen punto de partida para el estudio de nuevos modelos algorítmicos de EAs, ya que existen resultados demostrando su utilidad en la promoción de la difusión lenta de soluciones a través de la malla de la población y, por tanto, en el mantenimiento de la diversidad entre las soluciones que la componen [246]. Además, la investigación en cEAs es un campo con importantes avances recientes tanto en la teoría [115, 116] como en la práctica [28, 96].

En este capítulo ofrecemos primero algunas definiciones importantes en el marco de la optimización de problemas. Tras esto, presentamos en las posteriores secciones una breve introducción al campo de las metaheurísticas (Sección 2.2) y, en particular, a los algoritmos evolutivos en la Sección 2.3. En la Sección 2.4 describiremos los dos principales modelos existentes de descentralización de la población en EAs: los algoritmos evolutivos celulares y los distribuidos. Finalmente, profundizaremos en el principal caso de estudio de esta tesis, los algoritmos evolutivos celulares (Sección 2.5), prestando especial atención a la familia de cEAs llamada algoritmos genéticos celulares –cGAs– en la Sección 2.6. Este capítulo termina con una breve crítica al estado del arte actual de los cEAs, sugiriendo posibles líneas de trabajo que puedan mejorarlo. Estas sugerencias se tomarán como guía de las propuestas presentadas en esta tesis.

2.1. Optimización y Algoritmos Avanzados

En esta sección, definimos algunas nociones básicas que se utilizarán a lo largo de todo este documento. Comenzaremos dando una definición formal del concepto de optimización. Asumiendo el caso de la minimización, podemos definir un problema de optimización como sigue:

Definición 2.1 (Optimización) *Un problema de optimización se formaliza como un par (S, f), donde $S \neq \emptyset$ representa al espacio de soluciones –o de búsqueda– del problema, mientras que f es un criterio de calidad conocido como función objetivo, que se define como:*

$$f : S \rightarrow \mathbb{R} \ . \tag{2.1}$$

Así, resolver un problema de optimización consiste en encontrar un conjunto de valores adecuados de forma que la solución representada por estos valores $i^* \in S$ satisfaga la siguiente desigualdad:

$$f(i^*) \leq f(i), \quad \forall \ i \in S \ . \tag{2.2}$$

■

Asumir el caso de maximización o minimización no restringe en ningún caso la generalidad de los resultados, puesto que se puede establecer una igualdad entre tipos de problemas de maximización y minimización de la siguiente forma [37, 121]:

$$max\{f(i)|i \in S\} \equiv min\{-f(i)|i \in S\} \ . \tag{2.3}$$

En función del dominio al que pertenezca S, podemos definir problemas de *optimización binaria* ($S \subseteq \mathbb{B}$), *entera* ($S \subseteq \mathbb{N}$), *continua*($S \subseteq \mathbb{R}$), o *heterogénea* –o *mixta*– ($S \subseteq \{\mathbb{B} \cup \mathbb{N} \cup \mathbb{R}\}$).

Para resolver un problema de optimización, es necesario dar una definición de proximidad entre diferentes soluciones del espacio de búsqueda. Decimos que dos soluciones son próximas si pertenecen al mismo vecindario, y definimos el vecindario de una solución dada como:

Definición 2.2 (Vecindario) *Sea (S, f) un problema de optimización. Una estructura de vecindario en S se puede definir como:*

$$N : S \rightarrow S \ , \tag{2.4}$$

de forma que para cada solución $i \in S$ se define un conjunto $S_i \subseteq S$. Asumimos que si i es vecino de j, entonces j es también vecino de i: $j \in S_i$ sii $i \in S_j$.

■

En general, en un problema de optimización complejo como los considerados en este trabajo, la función objetivo puede presentar una solución óptima dentro de un vecindario determinado, pero que no lo es si consideramos el espacio de búsqueda completo. Por tanto, es fácil que un método de búsqueda pueda quedar atrapado en un valor óptimo dentro de un vecindario dado, pero que no coincida con la solución óptima al problema. Podemos definir, pues, un *óptimo local* como:

Definición 2.3 (Óptimo local) *Sea (S, f) un problema de optimización, y $S_{i'} \subseteq S$ el vecindario de una solución $i' \in S_{i'}$, i' es un óptimo local si se satisface la siguiente desigualdad:*

$$f(i') \leq f(i) \ , \quad \forall i \in S_{i'} \ . \tag{2.5}$$

■

A lo largo de nuestro trabajo nos enfrentaremos en algunas ocasiones a problemas de optimización en los que se deben satisfacer un conjunto de restricciones. De esta forma se restringe la región de soluciones admisibles S a aquellas que satisfagan todas las restricciones.

Definición 2.4 (Optimización con restricciones) *Dado un problema de optimización (S, f), definimos $M = \{i \in S | g_k(i) \geq 0 \ \forall \ k \in [1, \ldots, q]\}$ como la región de soluciones admisibles de la función objetivo $f : S \rightarrow \mathbb{R}$. Las funciones $g_k : S \rightarrow \mathbb{R}$ se denominan restricciones, y estas g_k se llaman de distinta forma según el valor que toma sobre $i \in S$:*

$$\begin{array}{lll} satisfecha & :\Leftrightarrow & g_k(i) \geq 0 \ , \\ activa & :\Leftrightarrow & g_k(i) = 0 \ , \\ inactiva & :\Leftrightarrow & g_k(i) > 0 \ , \ y \\ violada & :\Leftrightarrow & g_k(i) < 0 \ . \end{array}$$

El problema de optimización global se denomina sin restricciones sii $M = S$; en otro caso se le denomina restringido o con restricciones.

■

Nuestro trabajo se centra en la búsqueda del óptimo global al problema considerado, aunque debido a la dificultad de algunos problemas este criterio se puede ver relajado. A continuación, presentamos en las dos secciones que siguen una breve introducción a las metaheurísticas en general, y a los algoritmos evolutivos en particular, la familia de algoritmos objeto de estudio en esta tesis.

2.2. Resolución de Problemas con Metaheurísticas

En la literatura existen multitud de propuestas de técnicas algorítmicas, tanto exactas como aproximadas, para resolver problemas de optimización (vea la Figura 2.1 para una clasificación simple). Los algoritmos exactos garantizan que encuentran una solución óptima al problema para toda instancia de tamaño finito. Generalmente, los métodos exactos necesitan tiempos exponenciales de computación cuando tratamos con instancias grandes de problemas complejos. De manera muy clara, los problemas NP-difíciles no pueden abordarse de forma realista con técnicas exactas. Por tanto, el uso de técnicas aproximadas está recibiendo en las últimas décadas cada vez más atención. En estos métodos aproximados se sacrifica la garantía de encontrar el óptimo global al problema (en muchos casos, aunque no siempre) con el fin de poder encontrar soluciones buenas en un tiempo significativamente reducido en comparación con los métodos exactos.

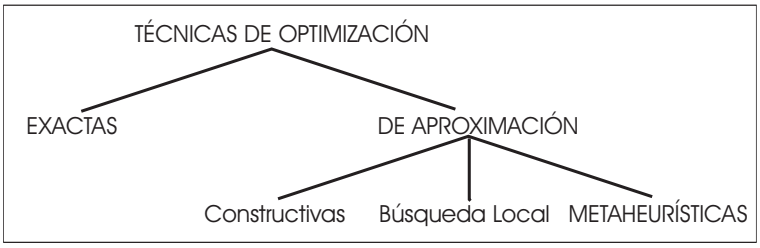


Figura 2.1: Clasificación de las técnicas de optimización.

En las dos últimas décadas ha emergido un nuevo tipo de técnicas aproximadas que consiste básicamente en la combinación de métodos heurísticos (técnicas aproximadas con componentes aleatorios guiados) básicos en entornos de más alto nivel con el fin de explorar el espacio de búsqueda de una forma eficiente y efectiva. Estos métodos son comúnmente conocidos con el término *metaheurísticas*. En [46] el lector puede encontrar recopiladas varias definiciones de metaheurística dadas por diferentes autores, pero en general podemos decir que las metaheurísticas son estrategias de alto nivel que planifican de manera estructurada la aplicación de varias operaciones para explorar espacios de búsqueda de elevada dimensión y complejidad intrínseca.

Las metaheurísticas pueden clasificarse en multitud de diferentes formas. En [46] se da una clasificación que depende de un conjunto de características seleccionadas que las diferencian. Así, podemos tener metaheurísticas *inspiradas en la naturaleza o no*, *basadas en poblaciones o en trayectorias*, *con función objetivo estática o dinámica*, que utilizan *una o varias estructuras de vecindario*, o que *memorizan estados anteriores de la búsqueda o no*. El lector puede encontrar en [46] una explicación detallada de esta clasificación. De entre las metaheurísticas más conocidas, podemos destacar a los algoritmos evolutivos (EA) [40], la búsqueda local iterada (ILS) [186], el enfriamiento simulado (SA) [162], la búsqueda tabú (TS) [120] y la búsqueda en vecindarios variables (VNS) [210].

2.3. Algoritmos Evolutivos

Alrededor de los años 60, algunos investigadores visionarios coincidieron (de forma independiente) en la idea de implementar algoritmos basados en el modelo de evolución orgánica como un intento de resolver tareas de optimización duras en ordenadores. Hoy en día, debido a su robustez, a su amplia

aplicabilidad, y también a la disponibilidad de una cada vez mayor potencia computacional, e incluso programas paralelos, el campo de investigación resultante, el de la computación evolutiva, recibe una atención creciente por parte de los investigadores de un gran número de disciplinas.

El marco de la computación evolutiva [40] establece una aproximación para resolver el problema de buscar valores óptimos mediante el uso de modelos computacionales basados en procesos evolutivos (algoritmos evolutivos). Los algoritmos evolutivos (EAs) son técnicas de optimización que trabajan sobre poblaciones de soluciones y que están diseñadas para buscar valores óptimos en espacios complejos. Están basados en procesos biológicos que se pueden apreciar en la naturaleza, como la selección natural [69] o la herencia genética [199]. Parte de la evolución está determinada por la selección natural de individuos diferentes compitiendo por recursos en su entorno. Por tanto, algunos individuos son mejores que otros. Es deseable que aquellos individuos que son mejores sobrevivan y propaguen su material genético.

La reproducción sexual permite el intercambio del material genético de los cromosomas, produciendo así descendientes que contienen una combinación de la información genética de sus padres. Éste es el operador de recombinación utilizado en los EAs, algunas veces llamado operador de cruce debido a la forma en que los biólogos han observado a las cadenas de los cromosomas cruzándose entre sí. La recombinación ocurre en un entorno en el que la selección de los individuos que tienen que emparejarse es principalmente una función del valor de fitness del individuo, es decir, de cómo de bueno es el individuo comparado con los de su entorno.

Como en el caso biológico, los individuos pueden sufrir mutaciones ocasionalmente. La mutación es una fuente importante de diversidad para los EAs. En un EA, se introduce normalmente una gran cantidad de diversidad al comienzo del algoritmo mediante la generación de una población de individuos aleatorios. La importancia de la mutación, que introduce aún más diversidad mientras el algoritmo se ejecuta, es objeto de debate. Algunos se refieren a la mutación como un operador de segundo plano, que simplemente reemplaza parte de la diversidad original que se haya podido perder a lo largo de la evolución, mientras que otros ven a la mutación como el operador que juega el papel principal del proceso evolutivo.

En la Figura 2.2 se muestra el esquema del funcionamiento de un EA típico. Como puede verse, un EA procede de forma iterativa mediante la evolución de los individuos pertenecientes a la población actual. Esta evolución es normalmente consecuencia de la aplicación de operadores estocásticos de variación sobre la población, como la selección, recombinación y mutación, con el fin de calcular una generación completa de nuevos individuos. A cada individuo se le asigna un valor representativo de su aptitud para el problema tratado mediante evaluaciones de la función de adecuación (o de fitness). Esta medida puede ser una función objetiva (un modelo matemático o una simulación por ordenador) o también una función subjetiva, en la que una persona pueda elegir las mejores soluciones frente a las peores. El criterio de terminación se fija normalmente en alcanzar un número máximo de iteraciones (programado previamente) del algoritmo, o encontrar la solución óptima al problema (o una aproximación a la misma) en caso de que se conozca de antemano.

Pasaremos ahora a analizar detalladamente el funcionamiento de un algoritmo evolutivo, cuyo pseudocódigo se muestra en el Algoritmo 2.1. Como se ha dicho ya con anterioridad, los algoritmos evolutivos trabajan sobre poblaciones de individuos, que son potenciales soluciones al problema. La población inicial está compuesta usualmente por individuos creados aleatoriamente, aunque también existe cierta tradición en el uso de técnicas de optimización (preferiblemente de poca carga computacional) para crear los individuos que formarán la población inicial, permitiendo así que el EA comience su ejecución sobre un conjunto de soluciones más prometedoras que en el caso de generarlas aleatoriamente. Tras la generación de la población inicial se calcula el valor de adecuación de cada uno de los individuos que la forman y el algoritmo entra en el bucle reproductor. Este bucle consiste en la generación de una nueva

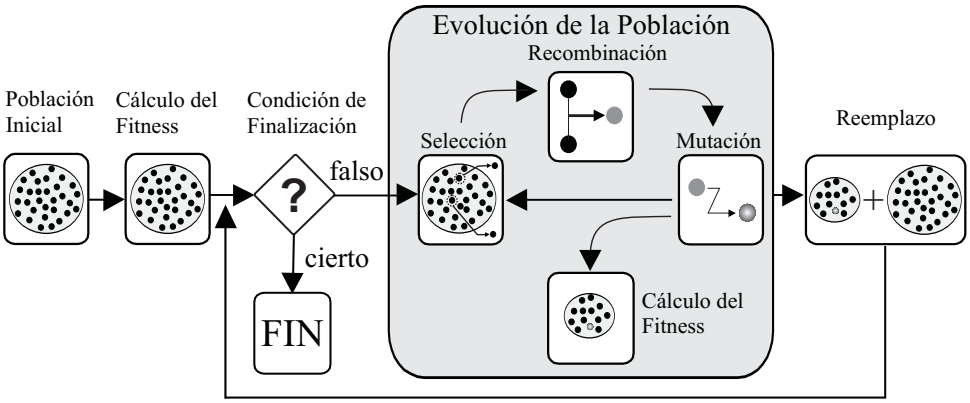


Figura 2.2: Funcionamiento de un EA típico.

población mediante la selección de los padres a cruzar, la recombinación de éstos, y la mutación de los descendientes obtenidos, tras lo que son evaluados. Esta nueva población generada por el bucle reproductor (P') se utilizará, junto con la población actual (P), para obtener la nueva población de individuos de la siguiente generación. El algoritmo devolverá la mejor solución encontrada durante la ejecución.

Podemos diferenciar dos tipos de EAs en función del tipo de reemplazo utilizado, es decir, en función de cómo se combina P' con P para generar la población de la nueva generación. Así, siendo μ el número de individuos de P y λ el de P', si la población de la nueva generación se obtiene de los mejores μ individuos de las poblaciones P y P' tenemos un $(\mu + \lambda)$-EA, mientras que si la población de la siguiente generación se compone únicamente de los μ mejores individuos de los λ que componen P', tendremos un (μ, λ)-EA. En este segundo caso, se cumple usualmente que $\mu \leq \lambda$.

La aplicación de EAs a problemas de optimización ha sido muy intensa durante la última década [40]. De hecho, es posible encontrar este tipo de algoritmos aplicado a la resolución de problemas complejos como tareas de optimización con restricciones, problemas con una función objetivo ruidosa, problemas con una elevada epistasis (alta correlación entre los valores a optimizar), y multimodalidad (existencia de numeroso óptimos locales) [15, 224]. La elevada complejidad y aplicabilidad de estos algoritmos ha promovido el surgimiento de nuevos modelos de optimización y búsqueda innovadores [76, 224].

Algoritmo 2.1 Pseudocódigo de un Algoritmo Evolutivo.

1. $P \leftarrow$ *GenerarPoblaciónInicial*();
2. *Evaluar*(P);
3. **mientras** !*CondiciónParada*() **hacer**
4. $P' \leftarrow$*SeleccionarPadres*(P);
5. $P' \leftarrow$*OperadoresDeVariación*(P');
6. *Evaluar*(P');
7. $P \leftarrow$ *SeleccionarNuevaPoblación*(P, P');
8. **fin mientras**
9. **Resultado:** La mejor solución encontrada

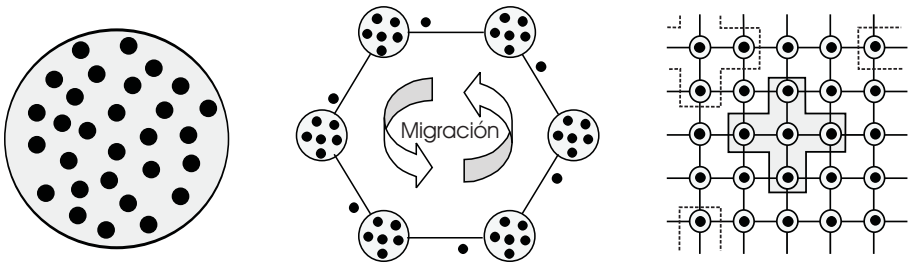


Figura 2.3: Un EA panmíctico tiene todos sus individuos –puntos negros– en la misma población (a). Si estructuramos la población podemos distinguir entre EAs distribuidos (b) y celulares (c).

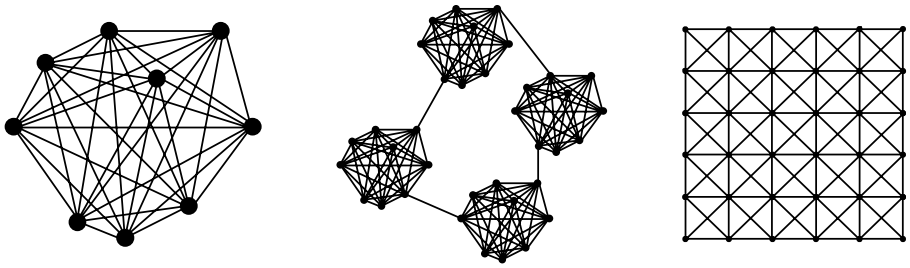
Fueron cuatro los primeros tipos de algoritmos evolutivos que surgieron [152]. Estas cuatro familias de algoritmos fueron desarrolladas simultáneamente por distintos grupos de investigación. Los algoritmos genéticos (GAs), fueron inicialmente estudiados por J. H. Holland [149, 150], en Ann Arbor (Michigan), H. J. Bremermann [49] en Berkeley (California), y A. S. Fraser [108] en Sidney (Australia). Las estrategias evolutivas fueron propuestas por I. Rechenberg [235, 236] y H.-P. Schwefel [251] en Berlin (Alemania), mientras que la programación evolutiva se propuso por primera vez por L. J. Fogel [102] en San Diego (California). Por último la cuarta familia de algoritmos, la programación genética, surgió dos décadas más tarde, en 1985, como una adaptación de N. Cramer [65] sobre un algoritmo genético que trabajaba con genes en forma de árbol, además de las cadenas de caracteres binarios utilizadas tradicionalmente en GAs.

En la actualidad, el campo de los algoritmos evolutivos se encuentra en pleno crecimiento y evolución. Prueba de ello son las nuevas familias de EAs que han surgido recientemente, como la optimización por enjambre de partículas (PSO) [47], la optimización por colonias de hormigas (ACO) [83], los algoritmos de estimación de distribuciones (EDAs) [176], o la búsqueda dispersa (SS) [173].

2.4. Algoritmos Evolutivos Descentralizados

La mayoría de los EAs trabajan sobre una única población (panmixia) de individuos, aplicando los operadores a la población como un todo (ver Figura 2.3a). Por otro lado, existe también una cierta tradición en el uso de EAs estructurados (en los que la población se descentraliza de alguna forma), especialmente en relación con su implementación en paralelo. El uso de poblaciones distribuidas de forma paralela en EAs se basa en la idea de que el aislamiento de poblaciones nos permite mantener una mayor diferenciación genética [295]. En multitud de casos [29], estos algoritmos con población descentralizada realizan un mejor muestreo del espacio de búsqueda y mejoran tanto el comportamiento numérico como el tiempo de ejecución de un algoritmo equivalente en panmixia. Entre los muchos tipos de EAs estructurados, los algoritmos *distribuidos* y los *celulares* son las herramientas de optimización más populares [27, 51, 195, 288] (ver Figura 2.3).

Por un lado, en el caso de los EAs distribuidos (dEAs), la población está particionada en un conjunto de islas en las que se ejecutan EAs aislados que intercambian algo de información durante la ejecución (ver Figura 2.3b). Este intercambio de información se realiza mediante la migración de individuos de unas poblaciones a otras con el fin de introducir algo de diversidad en cada una de estas sub-poblaciones, previniendo así a los algoritmos de cada isla de caer en óptimos locales de los que no puedan salir.



(a) (b) (c)

Figura 2.4: Grafos de conectividad entre individuos para EAs panmícticos, distribuidos y celulares.

Por otro lado, en el caso de los EAs celulares (cEAs), el concepto de *vecindario* (pequeño) se utiliza de una forma intensiva; esto significa que un individuo sólo puede interactuar con sus vecinos más cercanos en el ciclo reproductor (ver Figura 2.3c). Los pequeños vecindarios solapados de los cEAs ayudan en la exploración del espacio de búsqueda, ya que la lenta difusión de soluciones inducida en la población con los vecindarios dota al algoritmo de un tipo de exploración (diversificación), mientras que la explotación (intensificación) se da dentro de cada vecindario debido a la aplicación de las operaciones genéticas.

Si pensamos en la población de un EA en términos de grafos, siendo los vértices del grafo los individuos y los ejes las relaciones entre ellos, un EA panmíctico será un grafo completamente conectado (ver Figura 2.4a). Por otro lado, un cEA es un grafo en forma de retícula, de forma que un individuo sólo se puede relacionar con los más próximos (en 2.4c cada individuo tiene 8 vecinos), mientras que un dEA es una partición del EA panmíctico en varios EAs similares pero de menor tamaño, es decir, que en cada isla tenemos un grafo completamente conectado (convergencia muy rápida), mientras que existen muy pocas conexiones entre las islas.

Estos dos EAs descentralizados tradicionales (dEAs y cEAs) son dos subclases del mismo tipo de EA paralelo que consiste en un conjunto de sub-algoritmos que se comunican. Por tanto, las diferencias reales entre dEAs y cEAs se pueden establecer en la forma en que se estructura la población. En la Figura 2.5 mostramos una representación tridimensional (3D) de los algoritmos estructurados basada en el número de sub-poblaciones, el número de individuos contenido en cada una de ellas, y el grado de interacción que tienen estas sub-poblaciones [7]. Como se puede ver, un dEA está compuesto por unas pocas sub-poblaciones (con número de individuos $>> 1$), interaccionando poco entre sí. Por el contrario, los cEAs están compuestos por un alto número de sub-poblaciones altamente comunicadas, típicamente conteniendo un único individuo.

Este cubo se puede utilizar para obtener una forma generalizada para clasificar los EAs estructurados. De hecho, los puntos del cubo indicando el dEA y el cEA son únicamente "centroides", lo que significa que es posible encontrar o diseñar un algoritmo que difícilmente pueda ser clasificado dentro de estas dos clases de EAs estructurados ya que esta clasificación depende fuertemente de la selección de los valores en cada eje del cubo.

En un EA paralelo existen múltiples EAs elementales (granos) trabajando en sub-poblaciones separadas. Cada sub-algoritmo incluye una fase adicional periódica de comunicación con un conjunto de sub-algoritmos vecinos localizados en algún tipo de topología. Esta comunicación consiste usualmente en el intercambio de un conjunto de individuos o estadísticas de la población. Tradicionalmente, estos algoritmos fueron pensados de forma que todos los sub-algoritmos realizaran el mismo plan reproductor, aunque existe una tendencia reciente que consiste en la ejecución de EAs con distintas parametrizaciones en cada sub-población, realizándose de esta forma diferentes búsquedas sobre el espacio de soluciones en cada isla. Este tipo de EA paralelo se llama *heterogéneo* [23].

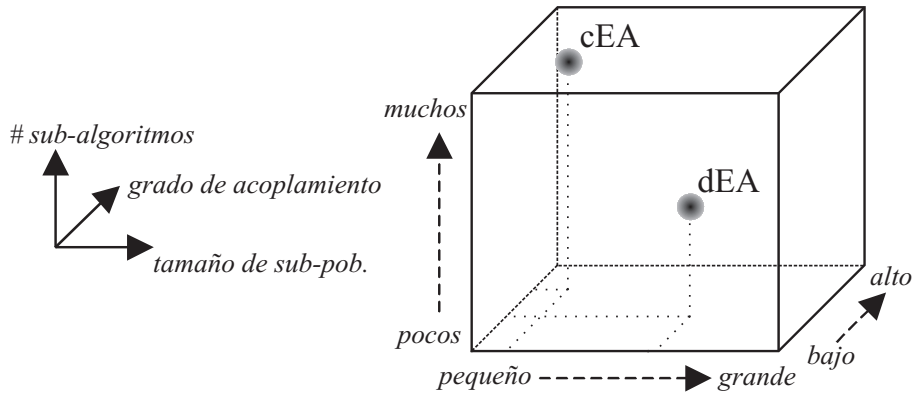


Figura 2.5: El cubo de los algoritmos evolutivos con población estructurada.

Desde este punto de vista, los EAs distribuidos y celulares sólo se diferencian en algunos parámetros, y pueden ser útiles incluso cuando se ejecutan en un único procesador [130]. Esto hace que mezclarlos sea una opción muy interesante, con el fin de obtener un algoritmo más flexible y eficiente para algunos tipos de aplicaciones [7, 190]. Además, cualquiera de estos EAs puede ser ejecutado en forma distribuida (son apropiados para un cluster de estaciones de trabajo).

2.5. Algoritmos Evolutivos Celulares

El modelo celular simula la evolución natural desde el punto de vista del individuo. La idea esencial de este modelo celular es proveer a la población de una estructura espacial que, como vimos en la sección anterior, puede definirse como un grafo conectado en el que cada vértice es un individuo que se comunica con sus vecinos más cercanos. En concreto, los individuos se encuentran dispuestos conceptualmente en una rejilla, y sólo se permite realizar la recombinación entre individuos cercanos. Esto nos lleva a un tipo de localidad conocido como *aislamiento por distancia*. El conjunto de emparejamientos potenciales de un individuo se llama *vecindario*. Se ha demostrado que en este tipo de algoritmos se pueden formar grupos de individuos similares, también llamados *nichos*, y que estos grupos funcionan de una forma que puede ser vista como subpoblaciones discretas (islas). De cualquier forma, no existen en realidad fronteras entre los grupos, y nichos cercanos pueden ser colonizados más fácilmente por nichos competidores que en un modelo de islas. Al mismo tiempo, nichos más distantes pueden ser afectados más lentamente.

En un cEA [273], la población está usualmente estructurada en una rejilla (o malla) bidimensional de individuos como la representada en la parte izquierda de la Figura 2.6, aunque usar esta topología no restringe el ámbito de las conclusiones obtenidas [246]. En ella, los individuos (círculos) situados al borde de la malla están conectados con los individuos que están en el otro borde de la malla en su misma fila y/o columna, según el caso. Esta conexión está representada en la figura mediante una línea discontinua. El efecto conseguido es el de una malla toroidal, de forma que todos los individuos tienen exactamente el mismo número de vecinos. Como se ha dicho anteriormente, la malla utilizada normalmente es bidimensional, aunque el número de dimensiones puede ser extendido fácilmente a tres o más.

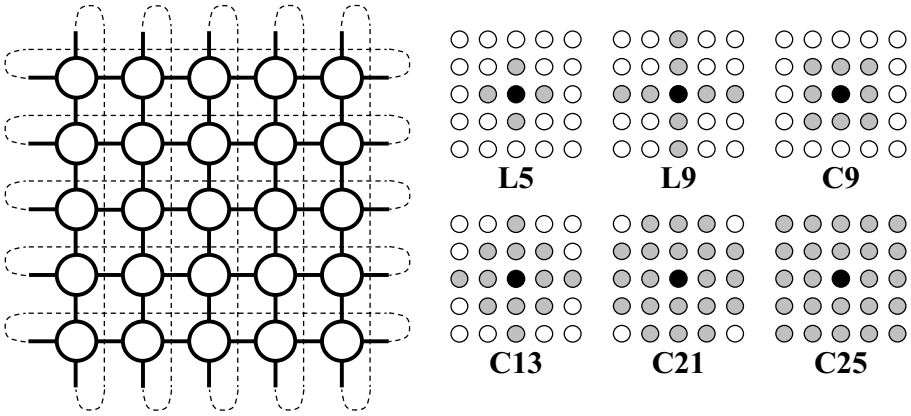


Figura 2.6: Población toroidal (izquierda) y vecindarios más típicamente utilizados (derecha) en cGAs.

El vecindario de un punto particular de la malla (donde hay alojado un individuo) se define en términos de la distancia de Manhattan entre el punto considerado de la malla y los otros en la población. Cada punto del grid tiene un vecindario que se solapa con los vecindarios de los individuos más próximos, y todos los vecindarios tienen tamaño y forma idénticos. En la parte derecha de la Figura 2.6 podemos ver los seis principales tipos de vecindarios típicamente utilizados en cGAs. En cuanto a los nombres de estos vecindarios, la etiqueta Ln (lineal) se utiliza para los vecindarios formados por los individuos vecinos que pueden ser alcanzados en $\leq n$ pasos tomados en una dirección axial determinada (norte, sur este u oeste), mientras que utilizamos la etiqueta Cn (compacto) para denominar los vecindarios que contienen los $n-1$ individuos más cercanos al considerado. Los dos vecindarios más comúnmente utilizados son: (i) L5 [195, 246, 288], también llamado vecindario de *NEWS* (por *North*, *East*, *West* y *South*), o de *Von Neumann*; y (ii) C9, al que también se le conoce como vecindario de *Moore*.

En cEAs, los individuos sólo pueden interactuar con sus vecinos en el bucle reproductor que aplica los operadores de variación. Este bucle reproductor es aplicado dentro del vecindario de cada individuo y consiste generalmente en seleccionar dos individuos del vecindario como padres de acuerdo a un cierto criterio, aplicarles los operadores de variación (recombinación y mutación), y reemplazar el individuo considerado por el descendiente recientemente creado siguiendo un determinado criterio, por ejemplo, si el descendiente representa una mejor solución que el individuo considerado. En la Figura 2.7 podemos ver cómo se aplica el ciclo reproductor en el ámbito de un individuo en un cEA (el vecindario considerado es NEWS). Según la política de actualización de la población utilizada, podemos distinguir entre cEAs síncronos y asíncronos (ver Sección 2.5.1).

El solapamiento de los vecindarios provee al cEA de un mecanismo implícito de migración, ya que las mejores soluciones se extenderán suavemente por toda la población, por lo que de esta forma logramos mantener la diversidad genética en la población por más tiempo con respecto a otros EAs no estructurados. Esta suave dispersión de las mejores soluciones por la población es una de las principales responsables del buen equilibrio entre exploración y explotación que los cEAs aplican en el espacio de soluciones al problema durante la búsqueda del óptimo. Es inmediato pensar que se puede regular esta velocidad de expansión de la mejor solución por la población (y por tanto el nivel de diversidad genética a lo largo de la evolución) modificando el tamaño del vecindario a utilizar, ya que el grado de solapamiento entre vecindarios crecerá con el tamaño del vecindario. En la Figura 2.8 se muestra el grado de

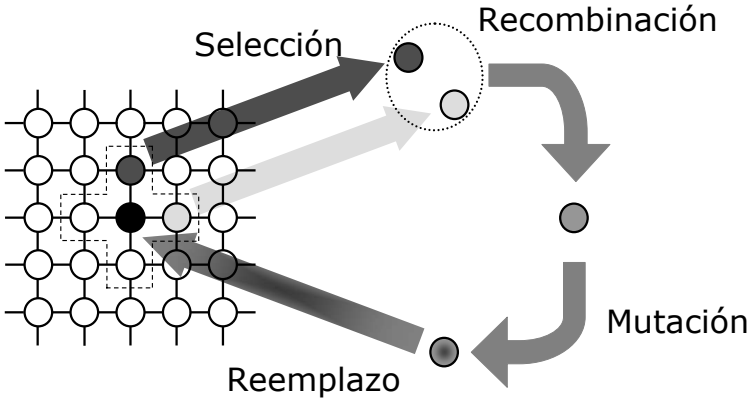


Figura 2.7: Ciclo reproductor de cada individuo en un cEA.

solapamiento de dos vecindarios distintos para los mismos individuos (coloreados con dos tonos distintos de degradado). En la parte de la izquierda se considera el vecindario NEWS, mientras que a la derecha de la figura se muestra el caso del vecindario C21. Los individuos sobre fondos del color gris más oscuro pertenecen a ambos vecindarios, mientras que los individuos sobre el fondo de color de los otros dos tonos de gris pertenecen únicamente a uno de los vecindarios: en el caso del gris más claro, pertenecen al vecindario del individuo coloreado con un degradado de grises claro, mientras que el otro tono de gris se emplea para mostrar los individuos pertenecientes al vecindario del individuo con degradado oscuro.

Un cEA puede verse como un autómata celular (CA) [293] con reglas de reescritura probabilísticas, en donde el alfabeto del CA es equivalente a todo el conjunto posible de cromosomas en el espacio de búsqueda [288, 272], es decir, al número de posibles soluciones al problema. Por tanto, si vemos a los cEAs como un tipo de CA, es posible importar herramientas analíticas y modelos o propuestas existentes en el campo de los CAs al de los cEAs con el fin de entender mejor a estos EA paralelos y poder mejorar su rendimiento.

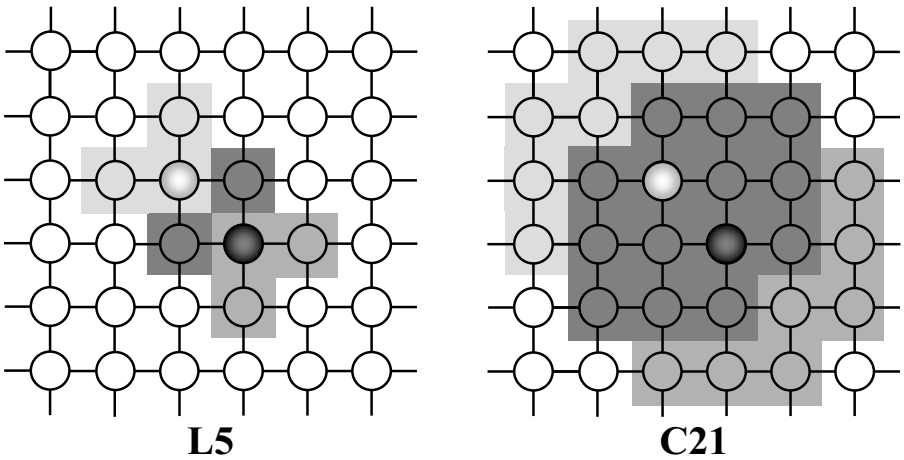


Figura 2.8: Vecindarios mayores inducen un mayor grado de migración implícita.

A continuación, se presentan en la Sección 2.5.1 los dos métodos de actualización de individuos que se pueden aplicar en cEAs, síncrono y asíncrono, así como las diferentes políticas de actualización existentes en el caso asíncrono. En la Sección 2.5.2 presentamos una medida para caracterizar cuantitativamente la rejilla y el vecindario de un cEA atendiendo a su "radio". Los conceptos presentados en estas dos secciones permitirán futuros estudios sobre políticas de actualización de individuos y tipos de malla y vecindarios.

2.5.1. Algoritmos Evolutivos Celulares Síncronos y Asíncronos

Existen dos tipos distintos de cEAs en función de cómo se aplique el ciclo reproductor a los individuos. Si el ciclo es aplicado a todos los individuos simultáneamente, decimos que el cEA es *síncrono*, puesto que los individuos que formarán la población de la siguiente generación son generados formalmente todos a la vez, de forma paralela. Por otro lado, si vamos actualizando los nuevos individuos en la población secuencialmente (individuos que formarán parte de la población de la siguiente generación) siguiendo alguna política determinada [20] en lugar de actualizarlos todos a la vez, tendremos un cEA *asíncrono*. Una excelente discusión acerca de autómatas celulares síncronos y asíncronos, que son esencialmente el mismo sistema que un cEA, está disponible en [250]. Como veremos a lo largo de esta tesis, la política de actualización de los individuos tiene un marcado efecto en el comportamiento del algoritmo, al igual que otros parámetros como el tamaño y forma del vecindario. Además, el orden de visita de los individuos en el caso asíncrono también es un factor determinante en el comportamiento del algoritmo. A continuación se presentan las principales políticas de actualización de individuos en cEAs asíncronos:

- **Barrido Lineal (Line Sweep – LS).** Este es el método más simple. Consiste en ir actualizando los individuos de la población secuencialmente por filas (1, 2, ..., n).
- **Barrido Aleatorio Fijo (Fixed Random Sweep – FRS).** En este caso, la siguiente celda a actualizar se selecciona con una probabilidad uniforme sin reemplazo (es decir, que no puede visitarse la misma celda dos veces en una generación); esto producirá una cierta secuencia de actualización $(c_1^j, c_2^k, \ldots, c_n^m)$, donde c_q^p significa que la celda número p se actualiza en el momento q y $(j, k, \ldots, m)$ es una permutación de las n celdas. La misma permutación se utiliza entonces para todas las generaciones.
- **Barrido Aleatorio Nuevo (New Line Sweep – NRS).** Es similar a FRS, con la diferencia de que se utiliza una nueva permutación aleatoria de las celdas en cada generación.
- **Elección Uniforme (Uniform Choice – UC).** En el caso de este último método, el siguiente individuo a visitar se elige aleatoriamente con probabilidad uniforme de entre todos los componentes de la población con reemplazo (por lo que se puede visitar la misma celda más de una vez en la misma generación). Esto se corresponde con una distribución binomial para la probabilidad de actualización.

Un *paso de tiempo* (o generación) se define como la actualización de n individuos secuencialmente, lo que se corresponde con actualizar todos los n individuos en los de la rejilla para LS, FRS y NRS, además del caso síncrono, y posiblemente menos de n individuos diferentes en el caso del método UC, ya que algunas celdas pueden ser actualizadas más de una vez en una misma generación. Es interesante destacar que, con la excepción de LS, las otras políticas de actualización asíncrona son estocásticas, lo que representa una fuente adicional de no determinismo más allá de la proporcionada por los operadores genéticos.

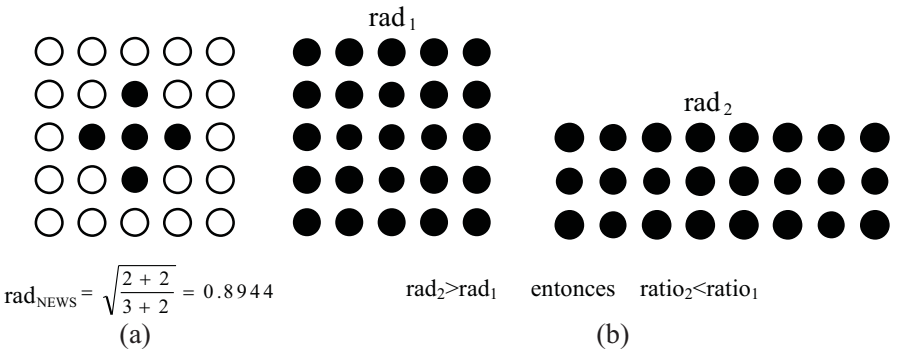


Figura 2.9: (a) Radio del vecindario NEWS. (b) Rejillas $5\times5=25$ y $3\times8\approx25$; mismo número de individuos con dos ratios diferentes.

2.5.2. Caracterización Formal de la Población en cEAs

En esta sección definimos los parámetros que caracterizan la población de un cEA. Utilizamos para ello la definición de *radio* presentada en [28], que es una extensión de la definición propuesta en [246] que tiene en cuenta el caso de rejillas no cuadradas. En [28] se considera que la rejilla tiene un radio igual a la dispersión de n^* puntos en un círculo centrado en $(\overline{x},\overline{y})$ (Ecuación 2.6). Esta definición siempre asigna diferentes valores numéricos a rejillas distintas, a diferencia de la definición propuesta en [246], que mide el radio del mínimo círculo que contiene a la rejilla, por lo que puede asignar valores iguales a rejillas distintas.

$$\text{rad} = \sqrt{\frac{\sum (x_i - \overline{x})^2 + \sum (y_i - \overline{y})^2}{n^*}} \ , \ \overline{x} = \frac{\sum_{i=1}^{n^*} x_i}{n^*} \ , \ \overline{y} = \frac{\sum_{i=1}^{n^*} y_i}{n^*} \ . \tag{2.6}$$

Esta definición no sólo caracteriza la forma de la rejilla sino que también puede proporcionar un valor de radio para el vecindario. Aunque se llama "radio", rad mide la dispersión de n^* patrones. Otras posibles medidas para vecindarios simétricos podrían asignar el mismo valor numérico a diferentes vecindarios, lo que no es deseable. Dos ejemplos son el radio del mínimo círculo rodeando un rectángulo que contenga al vecindario [246] o un *coeficiente de asimetría*.

Como se propuso en [246], la relación entre vecindario y rejilla puede cuantificarse mediante la relación (o *ratio*) entre sus radios (Ecuación 2.7). Los algoritmos que tienen una ratio similar muestran un comportamiento muy parecido en la búsqueda realizada cuando utilizan un mismo método de selección, como se concluye en [247].

$$\text{ratio}_{cEA} = \frac{\text{rad}_{\text{Vecindario}}}{\text{rad}_{\text{Rejilla}}} \ . \tag{2.7}$$

Si tenemos un número de individuos constante ($n = n^\star$ para hacer comparaciones justas), el valor del radio de la topología será mayor cuanto más estrecha sea la rejilla, como puede verse en la Figura 2.9b. Por tanto, si mantenemos constante el vecindario en tamaño y forma (por ejemplo, usamos NEWS, Figura 2.9a), la ratio será más pequeña cuanto más estrecha sea la rejilla de la población.

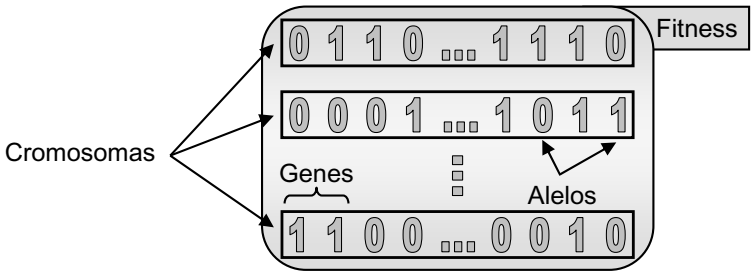


Figura 2.10: Ejemplo de la estructura de un individuo en un GA.

Una importante característica de la ratio es que reduciendo su valor logramos reducir la intensidad de selección global en la población, por lo que estamos promoviendo la exploración. Se espera que esto permita mantener una diversidad mayor en la población que pudiera mejorar los resultados obtenidos para problemas difíciles (como es el caso de las tareas multimodales o epistáticas). Además, la búsqueda realizada dentro de cada vecindario se encarga de guiar la explotación del algoritmo.

2.6. Algoritmos Genéticos Celulares

La familia de EAs que consideramos principalmente como caso de estudio en esta tesis es la de los algoritmos genéticos celulares. Los GAs codifican las variables de decisión de un problema de búsqueda en cadenas de variables de longitud finita de algún alfabeto de cierta cardinalidad. Las cadenas, que son soluciones candidatas al problema de búsqueda, son llamadas *cromosomas*, a cada una de las variables que forman el cromosoma se les llama *gen*, mientras que los distintos valores que pueden tomar estos genes se llaman *alelos*. En la Figura 2.10 puede verse un ejemplo de la estructura típica de un individuo de un GA.

Una vez que tenemos el problema codificado en uno o varios cromosomas y hemos definido una función de fitness para distinguir las soluciones mejores de las peores, podemos comenzar a evolucionar la población de soluciones al problema dado siguiendo los pasos que se dan a continuación:

1. **Inicialización.** La población inicial de soluciones candidatas se genera usualmente de forma uniformemente aleatoria, aunque se puede incorporar fácilmente conocimiento del problema u otro tipo de información en esta fase.

2. **Evaluación.** Una vez hemos inicializado la población, o cuando creamos una nueva solución descendiente, es necesario calcular el valor de adecuación de las soluciones candidatas.

3. **Selección.** Mediante la selección se trata de dar preferencia en la evolución a aquellas soluciones con un mayor valor de fitness, imponiéndose así un mecanismo que permite la supervivencia de los mejores individuos. La principal idea de la selección consiste en preferir las mejores soluciones frente a las peores, y existen muchos procedimientos de selección propuestos para cumplir con esta idea [61, 122, 247].

4. **Recombinación.** La recombinación combina partes de dos o más soluciones padres para crear nuevas soluciones (descendientes), que son posiblemente mejores. Existen muchas formas de conseguir esto, y un rendimiento competente del algoritmo depende de un mecanismo de recombinación bien diseñado. Idealmente, la solución descendiente que se obtiene de la recombinación no será idéntica a ninguno de los padres, sino que contendrá información combinada de los dos.

5. **Mutación.** Mientras la recombinación opera sobre dos o más cromosomas padre, la mutación modifica una única solución de forma aleatoria. De nuevo, existen multitud de variantes de mutaciones, que usualmente afectan a uno o más rasgos del individuo. En otras palabras, la mutación realiza un camino aleatorio en la vecindad de una solución candidata.

6. **Reemplazo.** La población descendiente que creamos mediante la selección, recombinación y mutación reemplaza a la población de padres siguiendo algún criterio determinado, como por ejemplo el reemplazo elitista, que preserva la mejor solución a lo largo de la evolución.

En el Algoritmo 2.2 presentamos el pseudocódigo de un cGA canónico. En él se puede ver como, tras la generación y evaluación de la población inicial, los operadores genéticos arriba mencionados (selección de los padres, su recombinación, la mutación del (los) descendiente(s) y el reemplazo del individuo actual por un descendiente) son aplicados a cada uno de los individuos dentro del entorno de sus vecindarios iterativamente hasta que se alcanza la condición de terminación. El pseudocódigo presentado en el Algoritmo 2.2 se corresponde con un cGA síncrono, puesto que los individuos que formarán parte de la población de la siguiente generación (bien los nuevos descendientes generados o bien los individuos de la población actual, dependiendo del criterio de reemplazo) van almacenándose en una población auxiliar que tras cada generación reemplaza a la población actual. Por tanto, en este modelo todos los individuos son actualizados simultáneamente en la población. En el caso asíncrono, no sería necesario utilizar la población auxiliar, ya que los descendientes generados reemplazan a los individuos de la población (según el criterio empleado) conforme van siendo visitados.

Algoritmo 2.2 Pseudocódigo de un cGA Canónico.

```
1.  proc Evoluciona(cga)        // Parámetros del algoritmo en 'cga'
2.  GeneraPoblaciónInicial(cga.pobl);
3.  Evaluación(cga.pobl);
4.  para s ← 1 hasta MAX_PASOS hacer
5.     para x ← 1 hasta cga.ANCHO hacer
6.        para y ← 1 hasta cga.ALTO hacer
7.           vecinos ← CalculaVecindario(cga,posición(x,y));
8.           padres ← Selección(vecinos);
9.           descendiente ← Recombinación(cga.Pc,padres);
10.          descendiente ← Mutación(cga.Pm,descendiente);
11.          Evaluación(descendiente);
12.          Remplazo(posición(x,y),pobl_auxiliar,descendiente);
13.       fin para
14.    fin para
15.    cga.pobl ← pobl_auxiliar;
16. fin para
17. fin proc Evoluciona
```

2.7. Críticas al Estado del Arte de los cEAs

El campo de los algoritmos evolutivos celulares tuvo una gran importancia con el auge de los computadores masivamente paralelos, para los que fueron especialmente diseñados, pero con la pérdida de popularidad que ha sufrido este tipo de computadores los cEAs fueron parcialmente olvidados por la comunidad científica. En los últimos años, gracias al trabajo de unos pocos grupos de investigadores conscientes de las ventajas que implica utilizar este tipo de algoritmos, han ido apareciendo poco a poco publicaciones poniendo de manifiesto el buen comportamiento de los cEAs en la resolución de problemas complejos sobre máquinas secuenciales [22, 28, 103, 106, 116]. Por tanto, los cEAs están, recibiendo cada vez más interés por parte de la comunidad científica, y son ya muchos grupos de investigación los que han puesto de manifiesto su interés en este tipo de algoritmos.

En los cEAs, como en el campo de la computación evolutiva en general, son muchas las líneas de investigación abiertas en la actualidad. En concreto, aún se ha hecho muy poco trabajo para entender bien el comportamiento de un EA. En 1991 aparece el primer trabajo en esta línea, según nuestro conocimiento. En él, Goldberg y Deb [122] proponen caracterizar el comportamiento de un algoritmo evolutivo mediante el tiempo de *takeover* (definido en el Capítulo 5). Tras este estudio, varios trabajos surgieron estudiando el tiempo de *takeover* y las curvas de crecimiento para cEAs [116, 117, 119, 135, 243, 247] y dEAs [24, 260]. En referencia al campo que nos ocupa, el de los cEAs, los trabajos existentes sólo han estudiado los casos simples de poblaciones lineales y cuadradas con distintas políticas de selección y de actualización de los individuos en la rejilla, y únicamente existe un estudio preliminar en el caso de las poblaciones de diversas formas rectangulares [119]. Como se demuestra a lo largo de este documento de tesis, la forma de la población es un factor determinante en el comportamiento de un cEA, hasta el punto de que cambiar la forma de la población puede llevarnos a diferencias en el comportamiento del algoritmo mucho mayores que en el caso de cambiar la política de actualización del individuo o el tipo de selección aplicado.

Una de las principales características que propician el buen comportamiento de los cEAs es la lenta difusión de las mejores soluciones por la población, por lo que se mantiene la diversidad de soluciones entre los individuos por más tiempo con respecto a otros tipos de EAs. Pero, probablemente, esta característica constituye uno de los principales problemas de los cEAs, ya que produce una lenta convergencia del algoritmo hacia el óptimo, disminuyendo de esta manera la eficiencia del algoritmo. De esta manera, son necesarios nuevos modelos algorítmicos que traten de beneficiarse de la lenta difusión de soluciones producida en la población de los cEAs pero que al mismo tiempo traten de mitigar la pérdida de eficiencia. En definitiva, han de surgir nuevos modelos que lleven a cabo sobre la población un equilibrio entre exploración y explotación más adecuado.

Existen múltiples familias de algoritmos evolutivos (la programación genética, los algoritmos genéticos, las estrategias evolutivas, la programación evolutiva, los algoritmos de estimación de distribuciones, etcétera), pero muy pocos cuentan con versiones celulares, por lo que se espera que diseñando versiones celulares para estos algoritmos se pueda mejorar su comportamiento.

Además, con la desaparición de los computadores masivamente paralelos, la mayoría de las nuevas implementaciones de cEAs publicadas en la literatura suelen ejecutarse en ordenadores secuenciales, y muy pocos son los modelos paralelos que se han propuesto desde entonces para aprovechar las características de las grandes y potentes redes de ordenadores existentes en la actualidad; algunos ejemplos son los publicados en [7, 105, 213]. También resultaría enormemente interesante la aportación de cEAs que puedan ejecutarse en sistemas de *grid computing*, para así sacar provecho de la utilización de las grandes redes de ordenadores distribuidas y heterogéneas que existen en la actualidad. En este campo quedan, por tanto, aún muchas aportaciones por realizar.

Por último, no existe ninguna adaptación ortodoxa del modelo celular al campo de la optimización multi-objetivo, y resulta interesante estudiar cuál sería el comportamiento de este tipo de algoritmos cuando es necesario optimizar más de un objetivo (usualmente en conflicto) simultáneamente.

2.8. Conclusiones

En este capítulo hemos presentado los algoritmos evolutivos, que son unos procesos iterativos que operan sobre un conjunto de individuos que forman una población; cada uno de estos individuos representa una solución potencial al problema. Esta población de individuos evoluciona gracias a la aplicación de un conjunto de operadores que están inspirados en algunos procesos biológicos de la naturaleza, como son la selección natural y la herencia genética. El efecto conseguido es que los individuos de la población son mejorados durante la evolución sin ser conscientes de ello. Los EAs son herramientas realmente útiles para la resolución de problemas complejos ya que trabajan rápidamente en grandes (y complejos) espacios de búsqueda gracias a que, al funcionar sobre una población de individuos, son capaces de operar sobre varias soluciones simultáneamente. Por tanto, los EAs pueden realizar varios caminos de búsqueda distintos a la vez.

Estructurando la población podemos mejorar el comportamiento numérico del algoritmo. Los principales tipos de EAs con población estructurada son los distribuidos y los celulares, aunque como se ha visto a lo largo de este capítulo, son únicamente dos de los múltiples tipos de EAs descentralizados existentes. El estudio del caso celular es el que nos ocupa en este trabajo de tesis, y es ampliamente explicado en este capítulo. El capítulo termina presentando los principales atributos que caracterizan a los algoritmos genéticos celulares.

Capítulo 3

El Estado del Arte en Algoritmos Evolutivos Celulares

Antes de comenzar cualquier investigación científica, es necesario en primer lugar conocer bien las aportaciones existentes en la literatura relativas al campo en el que se pretende investigar. Este paso de documentación es fundamental para el desarrollo de la ciencia, puesto que nos aportará un importante conocimiento del área de trabajo, lo que nos permitirá sacar provecho de las contribuciones de otros autores, y además evitaremos desarrollar estudios carentes de interés que, por ejemplo, ya hayan sido abordados con anterioridad. Por tanto, se ha realizado en este capítulo una amplia exploración del estado del arte en algoritmos evolutivos celulares, que incluye y clasifica algunas de las principales publicaciones existentes relacionadas con este campo.

La estructura del capítulo se detalla a continuación. Comenzamos en la Sección 3.1 explicando los primeros modelos de cEAs que fueron apareciendo en la literatura. La Sección 3.2 resume los principales estudios teóricos que se han desarrollado en cEAs, mientras que la Sección 3.3 recoge algunos de los trabajos más relevantes en los que se han llevado a cabo estudios empíricos del funcionamiento de cEAs, así como comparaciones con otros modelos. Un resumen de los trabajos más importantes que han aportado mejoras algorítmicas al campo de los cEAs se muestra en la Sección 3.4. La Sección 3.5 recopila algunos de los trabajos con mayor repercusión en el campo de los cEAs paralelos. Por último, al final del capítulo presentamos nuestras conclusiones.

3.1. EAs Celulares: un Nuevo Modelo Algorítmico

Los algoritmos evolutivos celulares fueron inicialmente diseñados para trabajar en maquinas masivamente paralelas, formadas por multitud de procesadores que ejecutan simultáneamente las mismas instrucciones sobre diferentes datos (máquinas SIMD – Single Instruction Multiple Data) [100]. En el caso más simple, los cGAs que se ejecutaban en este tipo de máquinas utilizaban una única población grande y asignaban un único individuo por procesador. Con el fin de evitar una elevada sobrecarga en las comunicaciones, el emparejamiento de los individuos fue restringido a los individuos más cercanos (es decir, a los pertenecientes a su vecindario).

En 1976, Bethke [44] realizó el estudio teórico de un GA sobre una máquina paralela SIMD, analizando la eficiencia de uso de la capacidad de procesamiento. Concluyó que el máximo de eficiencia se obtiene cuando se evalúan funciones de fitness mucho más costosas que las operaciones de evolución, un caso típico para muchas aplicaciones.

El primer modelo de cGA conocido es el propuesto por Robertson en 1987 [241], implementado sobre un equipo CM1, era un modelo en el que todas las fases del algoritmo (selección de progenitores, selección de población reemplazada, recombinación y mutación) se ejecutaban en paralelo. Su modelo reportó buenos resultados, con un tiempo de ejecución independiente del tamaño de la población.

Un año más tarde, Mühlenbein, Gorges Schleuter, y Kramer publicaron un trabajo [203] en el que proponían un cGA sobre máquinas masivamente paralelas para el problema TSP. Una importante característica de este cGA es que incorpora un paso de búsqueda local para tratar de mejorar las soluciones generadas por los operadores de recombinación y mutación. Se puede considerar, por tanto, que este algoritmo es el primer cGA híbrido publicado.

Tras estos dos primeros trabajos han aparecido después en pocos años varios cGAs, que fueron bautizados bajo los términos de modelo de *polinización de plantas* (*plant pollination model*) [121], *de individuos paralelos* (*parallel individual model*) [146], *de difusión* (*diffusion model*) [40], *de grano fino* (*fine grained*) [195], *masivamente paralelo* [144, 258] o *de selección local* [132]. Sin embargo, no fue hasta 1993 cuando Whitley dio pie por primera vez al término de GA celular en un trabajo en el que se aplica un modelo de autómata celular sobre un algoritmo genético [288].

Todos estos cGAs fueron inicialmente diseñados para trabajar en máquinas masivamente paralelas, aunque, debido a la rápida pérdida de popularidad que han sufrido este tipo de máquinas, el modelo fue adoptado más tarde para funcionar también en máquinas mono-procesador, sin ningún tipo de relación con el paralelismo. De hecho, desde la aparición de los cEAs, han existido implementaciones en entornos secuenciales [71], en redes de *transputers* [131], o en entornos paralelos distribuidos [196]. Este aspecto debe estar bien claro, ya que muchos investigadores aún piensan en la equivalencia entre GAs masivamente paralelos y GAs celulares, lo que representa un enlace erróneo hoy en día: los cGAs son simplemente una clase diferente de EAs, como lo son los algoritmos meméticos [211], los algoritmos de estimación de distribuciones [177, 204], o los algoritmos basados en enjambres de partículas [47].

Desde la aparición de los cGAs, muchas son las aportaciones que se han publicado en este campo. En la Tabla 3.1 resumimos algunas de las más importantes, que son comentadas en las secciones 3.2 a 3.4. Las aportaciones que se han realizado durante esta tesis doctoral se marcan en color gris.

3.2. La Investigación en la Teoría de los Modelos Celulares

Se ha hecho muy poco en el área de la teoría de los cGAs. Esto se puede deber en parte a la dificultad de deducir pruebas genéricas en un área en el que existen tantas posibilidades de implementación diferentes. Quizás, otra posible razón para esta carencia sea la creencia generalizada de que los cGAs modelan de una forma más correcta las poblaciones de la naturaleza que el modelo de islas o los GAs secuenciales. Independientemente de la razón de esta escasez de teoría, es necesario que se realice más trabajo en este área.

La Sección 3.2.1 presenta algunos estudios que tratan de modelar de forma teórica el comportamiento de los cGAs, mientras que la Sección 3.2.2 muestra un resumen de los principales trabajos que tratan de caracterizar el comportamiento de los cGAs en función de la ratio entre vecindario y población.

Tabla 3.1: Breve resumen de las principales aportaciones realizadas al campo de los cEAs.

Referencia		Autores	Aportaciones
(1990)	[144]	Hillis	cGA con dos poblaciones que co-evolucionan
(1991)	[61]	Collins & Jefferson	Estudio de la influencia de distintos métodos de selección
(1992)	[132]	Gorges-Schleuter	cGA con un mecanismo de migración
(1992)	[128]	Gordon et al.	cGA con un mecanismo de migración
(1993)	[287]	White & Pettey	Estudio de diferentes modos de aplicar la selección
(1995)	[244]	Rudolph & Sprave	Uso de un umbral de aceptación auto-adaptativo
(1996)	[246]	Sarma & De Jong	Primer estudio teórico de la presión de selección en cGAs
(1997)	[247]	Sarma & De Jong	Estudio de la influencia de distintos métodos de selección
(1997)	[255]	Sipper	cGA co-evolutivo
(1998)	[103]	Folino et al.	cGA con búsqueda local para SAT
(1998)	[178]	Laumanns et al.	Algoritmo depredador/presa para el dominio multi-objetivo
(1999)	[127]	Gordon et al.	cGA heterogéneo: distinta parametrización en cada celda
(1999)	[135]	Gorges-Schleuter	Comp. de ESs panmíctico y celular con topología en anillo y toroidal
(1999)	[166]	Kirley te al.	cGA que permite la existencia de celdas vacías (sin individuo)
(1999)	[172]	Ku, Mak & Siu	cGA con búsqueda local para entrenar redes neuronales recurrentes
(1999)	[260]	Sprave	Modelo basado en hipergrafos para caracterizar cEAs
(2000)	[28]	Alba & Troya	Cambia la forma de la población para regular la exploración/explotación
(2000)	[165]	Kirley & Green	cGA aplicado al dominio de la optimización continua
(2000)	[179]	Lee, Park & Kim	cGA con migración. Los individuos son mutados durante la migración
(2000)	[243]	Rudolph	Estudio del *takeover* en cGAs con población en anillo y toroidal
(2000)	[240]	Krink et al.	cGA con desastres. En las celdas vacías se colocan individuos nuevos
(2000)	[271]	Thomsen & Kirley	RBGA: cGA basado en religiones
(2001)	[170]	Krink et al.	[240] con un modelo de saco de arena para la frecuencia de desastres
(2001)	[185]	Llorà & Garrell	GALE: cGA para minería de datos. Permite celdas vacías
(2002)	[164]	Kirley	CGAD: cGA con desastres
(2003)	[13, 15]	Alba & Dorronsoro	cGAs con población adaptativa
(2002)	[183]	Li & Sutherland	Algoritmo depredador/presa para optimización continua
(2003)	[117]	Giacobini et al.	Estudio del takeover en cGAs asíncronos con población en anillo
(2003)	[116]	Giacobini et al.	Estudio de la presión de selección en cGAs con población en anillo
(2003)	[182]	Li	Algoritmo depredador/presa para el dominio multi-objetivo
(2004)	[11]	Alba et al.	Comparación entre cGAs y otros EAs
(2004)	[14]	Alba & Dorronsoro	Varios cGAs híbridos para VRP
(2004)	[20, 87]	Dorronsoro et al.	Estudio comparativo entre cGAs síncronos y asíncronos
(2004)	[115]	Giacobini et al.	Presión de selección en cGAs asíncronos con población toroidal
(2005)	[9]	Alba et al.	cGAs asíncronos con poblaciones adaptativas
(2005)	[10, 21]	Alba et al.	cMOGA: primer cGA multi-objetivo ortodoxo. Aplicado sobre MANETs
(2005)	[18]	Alba et al.	Primer cGA memético (cMA). Aplicado a SAT
(2005)	[26]	Alba & Saucedo	Comparación de cGA con GAs panmícticos
(2005)	[81]	Dick	Un cGA con población en anillo como método para preservar nichos
(2005)	[82]	Dick	Modelado matemático de la evolución genética en cGAs (pobl. en anillo)
(2005)	[119]	Giacobini et al.	Modelado de cGAs con población cuadrada y rectangular
(2005)	[118]	Giacobini et al.	Modelado de cGAs con poblaciones con topologías de mundo pequeño
(2006)	[16]	Alba & Dorronsoro	cGA híbrido que mejora el estado del arte en VRP
(2006)	[25]	Alba et al.	Primer EDA con población estructurada en forma celular
(2006)	[86]	Dorronsoro & Alba	cGA para el dominio de optimización numérica
(2006)	[140]	Grimme & Schmitt	Algoritmo depredador/presa para el dominio multi-objetivo
(2006)	[153]	Ishibuchi et al.	cGA con diferentes estructuras de vecindario para selección y cruce
(2006)	[189]	Luo & Liu	cGA reemplazando la mutación por búsqueda local. Diseñado para GPUs
(2006)	[187, 188]	Luna et al.	Comparación entre cMOGA y otros algoritmos MO del estado del arte
(2006)	[216, 217]	Nebro et al.	MOCell: un nuevo cGA MO ortodoxo
(2006)	[227]	Payne & Eppstein	Estudio de la topología de emparejamiento emergente en cGAs
(2006)	[154]	Janson et al.	HcGAs: cGAs con jerarquía en la población
(2006)	[254]	Simoncini et al.	Propuesta del nuevo operador de selección anisótropa para cGAs
(2006)	[297]	Xhafa	cMA para la planificación de tareas en *computación grid*
(2007)	[17]	Alba & Dorronsoro	Amplio estudio de un cGA memético sobre VRP
(2007)	[298]	Xhafa et al.	cMA para la planif. de tareas en *comput. grid*. Comp. con otros GAs
(2007)	[218]	Nebro et al.	Mejoras de MOCell y comparación con MOs del estado del arte
(2007)	[92]	jMetal	Biblioteca de algoritmos multi-objetivo. Incluye MOCell

3.2.1. Caracterizando el Comportamiento de cEAs

Existe una forma sencilla de caracterizar la búsqueda que realiza un cEA mediante la presión de selección, que es una medida de la velocidad con la que se difunden las buenas soluciones por la población. Con el fin de profundizar en el conocimiento del funcionamiento de los cEAs, algunos trabajos teóricos comparan los algoritmos en función de la presión de selección que muestran, e incluso en algunos casos se trata de modelar matemáticamente su comportamiento.

Sarma y De Jong hicieron en [246, 247] algunos estudios teóricos sobre la presión de selección inducida por cGAs con operadores de selección, tamaños y formas de vecindario diferentes. Para estudiar el efecto del tamaño del vecindario en la presión de selección, propusieron en [246] una definición del radio del vecindario como una medida de su tamaño. Además, observaron el mismo efecto al cambiar el tamaño de la población, por lo que propusieron una nueva medida, llamada *ratio*, que se define como la relación entre los radios del vecindario y la población. Sarma y De Jong descubrieron que esta ratio es un factor clave para controlar la intensidad de selección del algoritmo. Por tanto, dos algoritmos que tengan tamaños de población y vecindario distintos, pero con el mismo valor de la ratio tienen una presión de selección similar. Por último, propusieron el uso de una función logística (parametrizada con una variable) para aproximar la curva de la presión de selección que presentan los cGAs. Esta función se basa en la familia de curvas logísticas que se había demostrado en un trabajo anterior [122] que funcionan en el caso panmíctico. El modelo propuesto resultó ser una buena aproximación para cGAs con poblaciones cuadradas, pero más tarde se demostraron sus carencias en el caso de utilizar poblaciones rectangulares (consulte el Capítulo 5 para más información).

Tres años más tarde, Sprave propone en [260] una descripción unificada de cualquier tipo de EA con población tanto estructurada como no estructurada basada en el concepto de *hipergrafo*. Un hipergrafo es una extensión de un grafo canónico, en el que el concepto de ejes es generalizado: de la unión de un par de vértices pasan a ser uniones de subconjuntos de vértices.

Utilizando el concepto de hipergrafo, Sprave desarrolló en [260] un método para estimar la curva de crecimiento de la presión de selección de un GA. Este método se basa en el cálculo del diámetro de la estructura de la población y en la probabilidad de la distribución inducida por el operador de selección.

Gorges-Schleuter estudió en [135] las curvas de crecimiento para un modelo de difusión (celular) de Estrategias de Evolución (ES) con poblaciones estructuradas en forma de anillo o toroide. En sus estudios, observaron que el modelo de difusión de ES (tanto el modelo en anillo como en toroide) tiene una menor presión de selección que la ES equivalente con población panmíctica. Además, comparando ambos modelos de difusión, concluyeron que usando el mismo tamaño de vecindario, estructurar la población en anillo permite obtener una presión de selección menor que en el caso de utilizar una población toroidal.

En este mismo trabajo, se analizan también las diferencias en el comportamiento del algoritmo con diferentes esquemas de selección. En concreto, se estudian los casos en que se obliga a que uno de los padres sea el individuo actual o, por el contrario, sea elegido también por el mismo método de selección que el primero. También se estudia el efecto de permitir que el mismo individuo sea elegido como ambos padres (auto-emparejamiento) o no. El hecho de obligar a que el individuo actual sea uno de los padres permite reducir el error inducido por el muestreo estocástico, y además conduce a cambios más graduales de los individuos debido a que un descendiente que sobreviva reemplazará a dicho padre.

En [117], Giacobini et al. proponen modelos cuantitativos para estimar el tiempo de *takeover* (el tiempo que tarda la población en ser colonizada por copias del mejor individuo bajo los efectos de la selección únicamente) para cGAs síncronos y asíncronos con una población estructurada en forma de anillo (una dimensión), y utilizando un vecindario compuesto por los dos individuos más cercanos al

considerado. Este trabajo fue extendido más tarde en [115, 116] con el propósito de encontrar modelos matemáticos precisos para ajustar las curvas de la presión de selección de cGAs síncronos y asíncronos. En estos trabajos, la población se estructura en una malla bidimensional, pero se obliga a que tenga forma cuadrada. En [119], los mismos autores proponen una serie de recurrencias probabilísticas para modelar el comportamiento de la presión de selección de varios cGAs síncronos y asíncronos con poblaciones lineales y toroidales de población cuadrada para dos esquemas de selección distintos. Estudian también el caso de poblaciones rectangulares para los cGAs síncronos y asíncronos, pero en este caso sólo validan los experimentos sobre un único esquema de selección. En el Capítulo 5 se ha demostrado que el modelo no es del todo satisfactorio cuando se prueban otros esquemas de selección distintos del estudiado en [119].

Finalmente, Giacobini, Tomassini y Tettamanzi proponen en [118] modelos matemáticos para aproximar la curva de crecimiento de cGAs que trabajan sobre poblaciones en las que se ha definido la topología mediante un grafo aleatorio, o mediante *grafos de mundo pequeño*, en los que la distancia entre dos individuos cualesquiera es en general mucho menor que en el caso de las mallas regulares comúnmente utilizadas (se trata de grafos que no son ni regulares ni completamente irregulares [221]).

Por último, en [254], Simoncini et al. proponen un nuevo operador de selección para cGAs llamado la *selección anisótropa* (*anisotropic selection*) para ajustar la presión de selección del algoritmo. Este nuevo método consiste en permitir la selección de los individuos del vecindario con diferentes probabilidades en función del lugar que ocupen. De esta manera, los autores potencian la aparición de nichos en la población. Se estudia en este trabajo la presión de selección del algoritmo sobre varias formas de población, pero se hecha de menos la comparación del nuevo algoritmo con el cGA canónico.

En este trabajo presentamos una aportación al campo de la teoría de cGAs mediante el desarrollo de una ecuación matemática más precisa que las existentes para modelar las curvas de la presión de selección de cGAs con poblaciones rectangulares y cuadrada, y válida para diferentes métodos de selección.

3.2.2. La Influencia de la Ratio

En la literatura, existen resultados (como [206] para el caso de instancias grandes del problema TSP, o [33, 96, 130] para optimización de funciones) que sugieren, pero no analizan, que la forma de la malla de la población influye realmente en la calidad de la búsqueda que realiza el algoritmo. Unos años más tarde, Sarma y De Jong definieron en [246, 247] el concepto de la ratio, mediante la que cuantificaban la relación existente entre los tamaños de la población y el vecindario (definidos con el nombre de *radio*), y demostraron que cGAs con distintos vecindarios y poblaciones, pero con valores similares de la ratio, mostraban un comportamiento en la búsqueda similar.

Sin embargo, no fue hasta el año 2000 cuando Alba y Troya [28] publicaron el primer estudio cuantitativo de la mejora obtenida en el rendimiento de un cGA con mallas no cuadradas. En dicho trabajo, se analiza el comportamiento de varios cGAs con diferentes formas de malla sobre algunos problemas, concluyendo que el uso de mallas no cuadradas propicia un comportamiento muy eficiente de los algoritmos. Además, Alba y Troya redefinen en [28] el concepto de *radio* como la dispersión de un conjunto de patrones, ya que la definición dada por Sarma y De Jong [246] puede asignar el mismo valor numérico a vecindarios distintos, que es una propiedad indeseable. Finalmente, podemos encontrar otra contribución realmente importante en [28], que consiste en cambiar la forma de la población en un punto predeterminado de la ejecución para cambiar la relación entre exploración y explotación que aplica el algoritmo sobre el espacio de búsqueda. De esta manera, los autores sacan provecho del comportamiento diferente que muestran los cGAs con distintas formas de la población, y de una manera muy sencilla (y libre de carga computacional) cambian el comportamiento del algoritmo, a mitad de la ejecución, de forma que de un estado de explotación local se pase a fomentar la exploración del espacio de búsqueda o viceversa.

Como aportación a este área, hemos desarrollado en este estudio un nuevo modelo adaptativo en el que se cambia de forma automática la forma de la población para regular el equilibrio entre exploración y explotación.

3.3. Estudios Empíricos sobre el Comportamiento de cEAs

En esta sección se presentan algunos trabajos importantes que tratan de analizar el comportamiento de los cEAs, como la forma en que evolucionan los individuos de la población, o la complejidad que presentan en función de los operadores utilizados.

Collins y Jefferson caracterizan en su trabajo [61] la diferencia entre los GAs panmícticos y los cGAs en términos de varios factores, entre los que se encuentran la diversidad de genotipo y de fenotipo, la velocidad de convergencia, o la robustez del algoritmo, concluyendo que el emparejamiento local realizado en los cGAs "... *es más apropiado para la evolución artificial ...*" frente a los GAs con población panmíctica. Los autores demuestran en este trabajo que, para un problema concreto con dos óptimos, raramente un GA panmíctico encuentra las dos soluciones, mientras que el cGA sí que encuentra generalmente ambas soluciones. La razón es que, gracias a la lenta difusión de las mejores soluciones que se produce en el caso del cGA, la diversidad se mantiene por más tiempo en la población, formándose en ella pequeños *nichos*, o agrupaciones de individuos parecidos, representando distintas regiones de búsqueda del algoritmo. Este trabajo ha servido de inspiración a otros modernos trabajos en los que se utilizan cGAs como métodos para encontrar múltiples soluciones óptimas a problemas [81].

Davidor desarrolló en [70] un estudio sobre un cGA con una malla bidimensional y un vecindario de ocho individuos. En él, utilizaba selección proporcional (en función del valor de fitness) para ambos padres, creándose dos descendientes en la etapa de recombinación, y colocando ambos descendientes en el vecindario con una probabilidad que depende de su valor de fitness. Usando este modelo, descubrió que el cGA mostraba una rápida convergencia, aunque de forma localizada, por lo que se formaban en la población nichos de individuos con valores de fitness cercanos a los óptimos. Esta rápida (y localizada) convergencia no es sorprendente si consideramos que la selección es muy efectiva en poblaciones muy pequeñas. Por tanto, podemos concluir de este trabajo que un comportamiento característico de los cGAs es la formación de diversos nichos en la población, dentro de los cuales el ciclo reproductor tiende a potenciar la especialización de los individuos que lo componen (se fomenta la explotación dentro de estas zonas). De esta afirmación se desprende que el cGA mantiene diversos caminos de búsqueda hacia soluciones diferentes, ya que cada uno de estos nichos puede verse como un camino de explotación del espacio de búsqueda.

En el mismo congreso en el que Davidor presentó el trabajo antes comentado, Spiessens y Manderick [258] publicaron un estudio comparativo de la complejidad temporal de su cGA con respecto a un GA secuencial. Debido a que el paso de evaluación es dependiente del problema, lo ignoraron en sus estudios, y fueron capaces de demostrar que la complejidad del cGA se incrementa linealmente con respecto a la longitud del genotipo. Por el contrario, la complejidad de un GA secuencial se incrementa de forma polinomial con respecto al tamaño de la población multiplicado por la longitud del genotipo. Ya que un incremento en la longitud del individuo debería teóricamente ir acompañado de un incremento en el tamaño de la población, un incremento en la longitud de un individuo afectará por tanto al tiempo de ejecución de un GA secuencial, pero no al de un cGA. Además, en este artículo los autores deducen el número esperado de individuos para los casos de utilizar los métodos de selección comunes en GAs, mostrando que la selección proporcional es la que menor presión de selección muestra.

En esta línea, pasamos a analizar los resultados de los experimentos que unos años después Sarma y De Jong presentaron en [74]. En este trabajo, obtuvieron un resultado realmente importante para cualquier investigador interesado en desarrollar un cGA. En sus experimentos, compararon varios cGAs utilizando diversos esquemas de selección, y se dieron cuenta de que dos de las selecciones estudiadas se comportaban de distinta forma a pesar de tener presiones de selección equivalentes. Según los autores, "*estos resultados destacan la importancia de un análisis de la variación de los esquemas de selección. Sin este análisis, es posible caer en la trampa de asumir que los algoritmos de selección que tienen una presión de selección esperada equivalente producen un comportamiento de búsqueda similar*".

En 1994, Gordon et al. [126] estudian siete cGAs con distintos vecindarios sobre problemas de optimización continua y discreta. En su trabajo, concluyen que los vecindarios de mayor tamaño funcionan mejor con los problemas más simples pero, por el contrario, en el caso de los problemas más complejos da mejor resultado el uso de vecindarios más pequeños.

Capcarrère et al. definieron en [53] un conjunto de medidas estadísticas muy útiles para comprender el comportamiento dinámico de los cEAs. En ese estudio se utilizaron dos tipos de estadísticas: basadas en genotipo y en fenotipo. Las métricas basadas en el genotipo miden aspectos relacionados con los cromosomas de los individuos de la población, mientras que las basadas en el fenotipo tienen en cuenta las propiedades de adecuación del individuo, básicamente en el fitness.

Más recientemente, Alba et al. realizan en [22] un estudio comparativo del comportamiento de cGAs siguiendo modelos de actualización síncronos y asíncronos de los individuos en la población. Los resultados obtenidos apuntan a que los cGAs asíncronos ejercen una mayor presión de selección que los síncronos, por lo que convergen más rápido y, generalmente, encuentran la solución antes que los síncronos en los problemas menos complejos de los estudiados. Por el contrario, en el caso de los problemas más duros, los cGAs síncronos parecen ser los que ofrecen un mejor rendimiento, puesto que los asíncronos se quedan estancados en óptimos locales con mayor frecuencia.

En [96], Eklund realiza un estudio empírico para determinar el método de selección y la forma y tamaño del vecindario más apropiados para un GP celular. Las conclusiones fueron que tanto el tamaño como la forma del vecindario ideales dependían del tamaño de la población. En cuanto al método de selección, cualquiera de los estudiados se comporta bien, siempre que se fuerce la presencia de elitismo en la población. Además, descubrió que cuantas más dimensiones tenga una población mayor será la velocidad de dispersión de las buenas soluciones y, por tanto, mayor tendrá que ser el tamaño de la población para obtener un buen comportamiento.

Finalmente, en este documento aportamos a la teoría en cEAs nuevos y rigurosos estudios teóricos y prácticos de la presión de selección en cGAs asíncronos y síncronos con distintas formas de población (Capítulo 5) [20, 87] y estudios comparativos entre cEAs y EAs panmícticos (consulte el Capítulo 8 para una comparación en el campo de los EDAs [25] y el Capítulo 13 para el caso de GAs [17]).

3.4. Mejoras Algorítmicas al Modelo Canónico

En esta sección destacamos algunas otras publicaciones relevantes en el mundo de los cGAs y que no pertenecen directamente a ninguna de las secciones anteriores.

Rudolph y Sprave presentan en [244] un cGA síncrono con la población estructurada en una topología de anillo con un umbral de aceptación auto-adaptativo, que se utiliza como una forma de añadir elitismo al algoritmo. El algoritmo es comparado con un GA panmíctico, que resulta tener un rendimiento considerablemente peor que el cGA.

En [127, 129], los autores presentan un algoritmo heterogéneo llamado GA basado en terreno (TBGA o *Terrain-Based GA*). La idea de TBGA es que no requiere el ajuste de ninguno de sus parámetros por parte del programador. Esto se consigue definiendo un rango de valores para cada uno de los parámetros, que se dispersan a lo largo de los ejes de la población del cGA. Así, en cada posición de la población existe una combinación de parámetros distinta, siendo similares los parámetros de posiciones vecinas.

El algoritmo TBGA ha sido utilizado también para encontrar una buena combinación de parámetros para un cGA. Los autores presentan dos métodos para buscar una buena configuración, y se basan en el hecho de que TBGA almacena el número de veces que se encuentra el mejor individuo actual de la población en cada posición de la malla. La idea general es que aquella localización que ha encontrado un mayor número de veces el mejor individuo de la población debe tener una buena parametrización. En [129] Gordon y Thein concluyen que el algoritmo con los parámetros ajustados tal y como se acaba de describir tiene un rendimiento marcadamente mejor que TBGA y que un cGA ajustado manualmente.

En la literatura, se han publicado varios cGAs hibridizados con métodos de búsqueda local. Algunos ejemplos son los cGAs para entrenar redes de neuronas artificiales recurrentes para resolver el problema de la dependencia de largo plazo [172] o para resolver la función XOR [171], y los más recientemente propuestos para el problema SAT por Folino et al. [103, 104], y por Luo and Liu [189], en el que el operador de mutación es reemplazado por un paso de búsqueda local. Este último algoritmo tiene la peculiaridad de haber sido desarrollado para ejecutarse en la unidad de procesamiento gráfica (GPU) en lugar de utilizar la unidad central de proceso (CPU) del ordenador.

También se han propuesto algunos modelos en los que se introducen extinciones de individuos en determinadas zonas de la población. Por ejemplo, Kirley propone CGAD [164, 166], un cGA con perturbaciones que se caracteriza por la posibilidad de la ocurrencia de desastres que eliminan a todos los individuos que están localizados en una zona determinada de la población. CGAD fue probado con éxito en problemas de optimización numérica, dinámica, y multi-objetivo [163], convirtiéndose en la única aproximación multi-objetivo de un modelo celular. Otra propuesta similar es la de Krink et al. [170, 240], en la que con cierta frecuencia, se generan desastres en zonas de la población en la que los individuos son sustituidos por nuevos individuos. La frecuencia de estos desastres está controlada por un modelo de saco de arena (consulte [170] para más detalles).

Kirley también propuso, junto con Thomsen y Rickers, en [271] un EA basado en religiones RBEA, en el que se establecen en la población un conjunto de religiones. En este modelo, los individuos pertenecen a sólo una religión, y únicamente se relacionan con individuos pertenecientes a la misma religión, fomentando de esta manera la formación de nichos. Ocasionalmente, los individuos pueden convertirse a otra religión. En este trabajo, los autores demostraron que el nuevo RBEA mejoraba los resultados de un EA panmíctico y un cEA para un conjunto de funciones numéricas.

Aunque los cGAs tienen una migración implícita dada por el solapamiento de los vecindarios, algunos autores tratan de poner más énfasis en este factor añadiendo algún otro tipo de migración explícita adicional. Ejemplos de este tipo de algoritmos son los presentados en [128, 132], en los que se introduce la migración (copiando un individuo en otro lugar de la población en intervalos predefinidos) como una forma de permitir que nichos lejanos puedan interactuar. Este tipo de migración se utiliza también en [179], donde Lee et al. proponen además otra política de migración consistente en aplicar el operador de mutación a los individuos que migran. Otro tipo de migración se da en el anteriormente comentado CGAD [164], en el que las zonas extinguidas son repobladas por réplicas del mejor individuo. La migración se produce porque estas zonas que se quedan vacías son repobladas por los mejores individuos de la población. Por último, también se produce un tipo de migración implícita en el modelo de Alba y Troya [28] explicado en la Sección 3.2.2, ya que el cambio en la forma de la población implica una redistribución de parte de los individuos que la forman.

En [185] se propuso un nuevo cGA, llamado *GALE* para el problema de clasificación en minería de datos. La peculiaridad de *GALE* frente a un cGA canónico es que permite la existencia de celdas vacías en la malla. Por tanto, los descendientes serán colocados en las celdas vacías de sus vecindarios, y si no existieran celdas vacías, reemplazarían al peor individuo del vecindario. Otra peculiaridad de este modelo es la existencia de un *paso de supervivencia*, en el que se decide, en términos del fitness de cada individuo y sus vecinos, si los individuos se mantienen para la siguiente generación o no.

En 2002, Li y Sutherland presentaron en [183] una variante de cGA llamada algoritmo de depredador/presa en el que las presas (que se corresponden con los individuos representando soluciones potenciales al problema) se mueven libremente por las posiciones de la malla, emparejándose con las presas vecinas en cada generación. Además, existe un número de depredadores que están continuamente desplazándose por la población, de forma que en cada generación matan a la presa más débil de su vecindario. El algoritmo mostró buenos resultados al ser comparado con un GA panmíctico y un GA heterogéneo distribuido (ambos fueron tomados de [142]) para un conjunto de 4 problemas numéricos. Este algoritmo fue más tarde extendido en [182] al dominio multi-objetivo con buenos resultados. De hecho, existen dos propuestas más de algoritmos depredador/presa para problemas multi-objetivo en la literatura, publicadas en [178] y [140].

Otro tipo de cGA no estándar es el dado por las aproximaciones *co-evolutivas*. El ejemplo más conocido es el del método de Hillis [144] para encontrar la red de ordenación mínima. La propuesta de Hillis consistía en un GA masivamente paralelo en el que existen dos poblaciones independientes, que evolucionan según un GA estándar. En una población, los *anfitriones* representan redes de ordenación, mientras que en la otra población los *parásitos* representan casos de prueba. El valor de fitness de las redes de ordenación viene dado por cómo ordenan los casos de prueba que proveen los *parásitos* en la misma localización de la malla. Por otro lado, los *parásitos* se puntúan según el número de desperfectos que pueden encontrar en la red de ordenación correspondiente (en la misma localización de la malla de *anfitriones*). De esta manera, el algoritmo evoluciona para tratar de encontrar la solución al problema, puesto que en la población de *parásitos* evolucionan los individuos hacia casos de prueba más difíciles, mientras que en la otra población evolucionan las redes de ordenación para resolver estos casos de estudio cada vez más difíciles.

Otra variación co-evolutiva interesante del modelo de cGA es el *algoritmo de programación celular* de Sipper [255]. La programación celular ha sido ampliamente utilizada para hacer evolucionar autómatas celulares para realizar tareas computacionales y está basado en la co-evolución topológica de reglas de autómatas celulares vecinas.

Para terminar esta sección, presentaremos brevemente los principales trabajos existentes en los que se han aplicado cGAs sobre problemas dinámicos de optimización. Algunos ejemplos son el trabajo de Kirley y Green [165], el ya mencionado CGAD [164] (de Kirley también), o el estudio comparativo del rendimiento de GAs panmícticos (de estado estacionario y generacional) y celulares realizado por Alba y Saucedo [26], del que se desprende que el cGA es, en general, el mejor de los tres algoritmos (junto con el panmíctico de estado estacionario en algunos casos).

Aunque podemos encontrar algunas propuestas en la literatura de cEAs aplicados al campo multi-objetivo, no existe ningún modelo ortodoxo. En este trabajo presentamos algunos nuevos modelos de cGA multi-objetivo que son adaptaciones del modelo de cGA. También se presenta en este trabajo un nuevo modelo de cGA con población jerárquica, llamado HcGA, en el que se potencia la explotación de las mejores soluciones, a la vez que se mantiene la diversidad en la población.

Tabla 3.2: Breve resumen de los principales modelos paralelos de cEAs.

Algoritmo	**Referencia**		**Modelo**
ASPARAGOS	[195]	(1989)	Asíncrono. Aplica una búsqueda local si no se produce mejora
ECO-GA	[70]	(1991)	Malla 2-D. Vecindario de 8 individuos. Obtiene dos descendientes
HSDGA	[284]	(1992)	GA jerárquico de grano fino y grueso
fgpGA	[33]	(1993)	cGA con dos individuos por procesador
GAME	[261]	(1993)	Biblioteca genérica para la construcción de modelos paralelos
PEGAsuS	[239]	(1993)	Grano fino o grueso. Mecanismo de programación para MIMD
LICE	[259]	(1994)	Modelo celular de ES
SPL2	[265]	(1994)	Grano fino o grueso. Muy flexible
Juille & Pollack	[158]	(1996)	Modelo celular de GP
dcGA	[64]	(1998)	Modelo de islas celulares o de estado estacionario
Gorges-Schleuter	[135]	(1999)	Modelo celular de ES
CAGE	[104]	(2001)	Modelo celular de GP
MALLBA	[93]	(2002)	Biblioteca genérica para la construcción de modelos paralelos
Combined cGA	[214]	(2003)	cGA con población dividida en varias sub-poblaciones celulares
ParadisEO	[50]	(2004)	Biblioteca genérica para la construcción de modelos paralelos
Weiner et al.	[285]	(2004)	modelo celular de ES con una estructura del vecindario variable
Meta-cGA	[190]	(2005)	cGA paralelo en redes de área local usando MALLBA
PEGA	[88]	(2007)	cGA distribuido en islas celulares para entornos de *computación grid*

3.5. Modelos Paralelos de cEAs

Como ya se ha comentado varias veces, los cEAs fueron inicialmente desarrollados en máquinas paralelas de memoria compartida, aunque también han surgido otros modelos más apropiados para las arquitecturas distribuidas que existen en la actualidad. En la Tabla 3.2 se muestra un resumen de los principales modelos paralelos de cEAs existentes en la literatura.

Ejemplos de cGAs que fueron desarrollados sobre máquinas SIMD fueron los estudiados por Manderick y Spiessens [195] (más tarde mejorado en [258]), Mühlenbein [201, 203], Gorges-Schleuter [131], Collins [61] y Davidor [70], en los que se colocan los individuos en una malla, restringiendo la selección y la recombinación a pequeños vecindarios en la malla. ASPARAGOS, el modelo de Mühlenbein y Gorges-Schleuter, fue implementado sobre una red de *transputers*, con la población estructurada en una escalera cíclica. Luego evolucionó incluyendo nuevas estructuras y mecanismos de emparejamiento [133] hasta constituirse en una herramienta efectiva de optimización [134].

Destacaremos también los trabajos de Talbi y Bessière [268], en el que estudian el uso de vecindarios de pequeño tamaño, y el de Baluja [33], en el que se analizan tres modelos de cGAs y un GA distribuido en islas sobre una máquina MasPar MP-1, obteniendo como resultado el mejor comportamiento de los modelos celulares. Sin embargo, Gordon y Whitley presentaron en [130] un estudio comparando un cGA con un GA de grano grueso, siendo los resultados de éste último ligeramente mejores. En [169] se muestra una comparación de varios cGAs con respecto al GA equivalente secuencial, mostrándose claramente las ventajas de utilizar el modelo celular. Podemos encontrar en [29] una comparación mucho más exhaustiva que las anteriores entre un cGA, GAs de estado estacionario y generacional, y un GA distribuido en un modelo de islas en términos de la complejidad temporal, la presión de selección, la eficacia y la eficiencia, entre otros aspectos. Los autores concluyen que existe una clara superioridad de los algoritmos estructurados frente a los no estructurados.

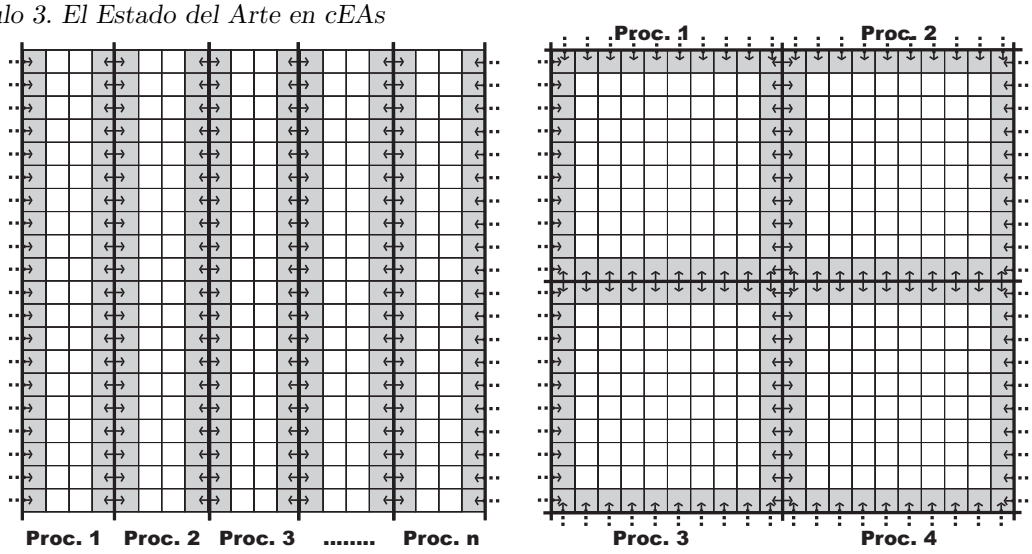


Figura 3.1: CAGE (izquierda) y el modelo de cGA paralelo combinado (derecha).

En el año 1993, Maruyama et al. proponen en [196] una versión de cGA sobre un sistema de máquinas en una red de area local con memoria distribuida. En este algoritmo, llamado DPGA, se coloca un individuo en cada procesador y, con el fin de reducir las comunicaciones al mínimo, en cada generación cada procesador envía una copia de su individuo a otro procesador elegido aleatoriamente. En cada procesador existe una lista de individuos *suspendidos*, que es donde se colocan los individuos que llegan de otros procesadores. A la hora de aplicar los operadores genéticos en cada procesador, esta lista de individuos *suspendidos* hará las veces de vecindario. Este modelo es comparado con un APGA, un cGA asíncrono propuesto por Maruyama et al. [197], un GA secuencial y un heurístico especializado para el problema tratado. Como conclusión, los autores destacan que DPGA muestra un rendimiento similar al del algoritmo secuencial equivalente.

Existen también modelos de cGAs paralelos más modernos, que funcionan sobre ordenadores conectados en redes de área local. Estos modelos deben ser diseñados de forma que se reduzcan al mínimo las comunicaciones puesto que, por sus propias características, los modelos celulares necesitan un elevado número de comunicaciones.

En este marco, Nakashima et al. proponen en [213] un *cGA combinado* en el que existen varias sub-poblaciones con estructura celular evolucionando, y que interaccionan entre sí por medio de sus bordes. Un gráfico de este modelo se puede ver en la parte derecha de la Figura 3.1. En un trabajo posterior [214], los autores proponen varias parametrizaciones con diferentes números de sub-poblaciones, formas de reemplazo y topologías de las sub-poblaciones, y analizan los resultados. Los autores utilizaron este modelo en una máquina secuencial, pero es directamente extrapolable a un modelo paralelo, en el que cada procesador contendría una de las sub-poblaciones.

Folino et al. proponen en [105] CAGE, un GP paralelo. En CAGE, la población se estructura en una malla toroidal de dos dimensiones, y se parte en grupos de columnas que formarán sub-poblaciones (ver el gráfico de la izquierda en la Figura 3.1). De esta forma, se reduce el número de mensajes que es necesario enviar con respecto a otros modelos que hagan divisiones de la población en dos dimensiones (ejes x e y). En CAGE, cada procesador contiene un número determinado de columnas que hace evolucionar y al final de la generación envía las columnas que están en los extremos a los procesadores vecinos, para que puedan utilizar esos individuos como vecinos de los individuos que están en los extremos de su sub-población.

Por último, existen también varias bibliotecas genéricas de programación de algoritmos paralelos que ofrecen facilidades para implementar cualquier tipo de algoritmo paralelo. Algunas de estas bibliotecas son GAME [261], ParadisEO [50] o MALLBA [93].

Para terminar, mencionamos la contribución que aportamos en esta tesis al campo de los cGAs paralelos. En el Capítulo 10 presentamos el meta-cGA, que ha sido desarrollado empleando la biblioteca MALLBA, y PEGA, un nuevo GA distribuido en islas celulares que puede ser ejecutado en entornos de red de área local o en *computación grid*. PEGA se ha aplicado a las instancias más grandes existentes del problema VRP, aportando soluciones nuevas al estado del arte.

3.6. Conclusiones

Hemos explorado en este capítulo la mayoría de los trabajos existentes en el campo de los algoritmos evolutivos celulares. Los trabajos analizados comprenden tanto las principales publicaciones del campo como las tendencias más recientes que están apareciendo. Este estudio nos permite adquirir un conocimiento del dominio de cEAs que será necesario para proponer y justificar las investigaciones desarrolladas en este trabajo de tesis.

Capítulo 4

Diseño Algorítmico y Experimental

Tras la introducción al campo de los cEAs y el estudio del estado del arte en ese dominio realizados en los capítulos anteriores, nos planteamos en este capítulo, como hipótesis, que es posible (y consideramos que muy interesante) aportar numerosas mejoras al modelo canónico de los cEAs con el fin de incrementar su rendimiento.

Para estudiar el comportamiento de los cEAs, necesitamos en primer lugar caracterizarlos formalmente (Sección 4.1). Una vez hayamos definido matemáticamente un cEA canónico, pasaremos a describir nuestra propuesta de solución para la hipótesis definida en la Sección 4.2. En la Sección 4.3 veremos el procedimiento de trabajo que hemos seguido en nuestros estudios para obtener relevancia estadística en nuestras conclusiones cuando comparamos los diferentes modelos. Para terminar, presentaremos en la Sección 12.3 nuestras principales conclusiones.

4.1. Formalización de GA Descentralizado

En esta sección presentamos una caracterización generalizada de los GAs descentralizados en general. Para ello, presentamos a continuación algunos conceptos que nos resultarán útiles partiendo de la definición de *sistema adaptativo* (AS), propuesta en [67].

Definición 4.1 (Sistema Adaptativo) *Definimos un* sistema adaptativo *como una tupla*

$$\mathcal{S} = (A, \vartheta, \tau, B, \vartheta_B, \tau_B, E) \ , \tag{4.1}$$

donde:

- $A = \{A_1, A_2, \ldots\}$ *es el* conjunto de estructuras *con las que trabaja* $\mathcal{S}$.
- $\vartheta = \{\omega_1, \omega_2, \ldots\}$ *es un* conjunto de operadores *para modificar estructuras de* A.
- $\tau : E \rightarrow \vartheta$ *es un* plan adaptativo *para determinar qué operador aplicar dada la entrada* E *en un instante determinado.*
- $B = \{B_1, B_2, \ldots\}$ *es el* conjunto de artefactos *manejados por* $\mathcal{S}$.
- $\vartheta_B = \{\vartheta_{B1}, \vartheta_{B2}, \ldots\}$ *es un* conjunto de operadores *para modificar artefactos en* B.

- $\tau_B : E \rightarrow \vartheta_B$ *es un* plan adaptativo *para determinar qué operador aplicar para modificar estructuras de* B *dada la entrada* E *en un instante determinado.*
- E *es el* conjunto de posibles entradas *al sistema (problema que se quiere resolver).* ■

Como se puede apreciar, existe una gran similitud entre los tres primeros elementos de $\mathcal{S}$ (A, ϑ y τ) y los tres siguientes (B, ϑ_B y τ_B), la diferencia entre ambas ternas radica en el dominio al que se aplican: el genotipo $\mathcal{G}$ en el primer caso y el fenotipo $\mathcal{P}$ en el segundo. Téngase en cuenta que los genotipos son representaciones internas de los fenotipos.

Con la Definición 4.1 no nos es suficiente aún para poder dar una descripción unificada de GA descentralizado. Para ello, sería necesario incluir en dicha definición el concepto de *granularidad* del algoritmo. Un AS granulado (GAS) se encuentra estructurado internamente en unidades estructurales (*gránulos*) comunicantes. Combinando adecuadamente dichas unidades estructurales obtenemos un algoritmo.

Definición 4.2 (Sistema Adaptativo Granulado) *Se define un* sistema adaptativo granulado *como una tupla*

$$\Xi = (\Delta, \Pi, \xi) \ , \tag{4.2}$$

donde:

- $\Delta = \{\Delta_1, \Delta_2, \ldots, \Delta_m\}$ *es un* conjunto de gránulos*, cada uno de ellos* (Δ_i) *consistente en un sistema adaptativo* $(A^i, \vartheta^i, \tau^i, B^i, \vartheta^i_B, \tau^i_B, E^i)$.
- Π *es la* política de activación *de gránulos.*
- $\xi : \Delta \times \Delta \rightarrow \mathcal{P}(B)$ *es una función que controla la* comunicación entre gránulos *(qué artefactos son compartidos entre ellos).* ■

Por lo tanto, un GAS está compuesto por varios gránulos, una política de activación de dichos gránulos (que puede ser secuencial Δ_i, Δ_j o concurrente $\Delta_i \| \Delta_j$), y una función de comunicación entre ellos. Cada uno de estos gránulos es, a su vez, un sistema adaptativo que utiliza sus propias estructuras y lleva a cabo su propio plan adaptativo.

La plantilla de un GAS es suficiente para realizar un cómputo arbitrario [37]. Pero el principal inconveniente de este modelo es su gran generalidad y, aunque permite definir los principales esquemas paralelos que pueden utilizarse a la hora de diseñar una metaheurística paralela, no introduce en su formulación los detalles más específicos, con lo que es difícil caracterizar ciertos aspectos algorítmicos muy concretos, como por ejemplo el sincronismo en las comunicaciones u operaciones de variación que involucran a estructuras de diferentes sistemas granulados.

Pasamos ahora a caracterizar un GA de población única centralizado a partir de la formalización de GAS dada en la Definición 4.2.

Definición 4.3 (Algoritmo Genético Centralizado) *Podemos definir un* GA centralizado *como un sistema adaptativo granulado* $\Xi_{\text{cent}} = (\Delta_{\text{cent}}, \Pi_{\text{cent}}, \xi_{\text{cent}})$ *de una sola tupla* $\Delta_{\text{cent}} = \{\Delta\}$*, y con* $\Pi_{\text{cent}} = \emptyset$ *y* $\xi_{\text{cent}} = \emptyset$*, puesto que sólo existe un gránulo. Para distinguir las variantes de estado estacionario y generacional, necesitamos especificar el contenido del gránulo:*

$$\Delta = (A, \vartheta, \tau, B, \vartheta_B, \tau_B, E) \ , \tag{4.3}$$

donde:

- $A = \{A_1, A_2, \ldots\}$ *es el* conjunto de poblaciones posibles. *De forma que cualquier población generada perteneciente a la secuencia de poblaciones* $P(i) = \bigcup_{t=0}^{\tau_{AE}} \Psi^t(P(0))$ *también pertenece a* A*, es decir,* $\exists\, i, j \;\; P(i) = A_j$.
- $\vartheta = \{\omega_{\text{cent}}\}$. *Existe un único operador, denominado* ciclo reproductor central. *Este operador se define por la concatenación de cuatro operadores internos:*

$$\omega_s \equiv s_{\Theta_s} \qquad \omega_c \equiv \otimes_{\Theta_c} \qquad \omega_m \equiv m_{\Theta_m} \qquad \omega_r \equiv r_{\Theta_r}$$

$$\omega_{\text{cent}}(A_i, E_i) = \omega_r(A_i \cup \{\omega_m(\vec{a}_{ij}) | \vec{a}_{ij} \in \omega_c(\omega_s(A_i, E_i))\})$$

- $\tau : E \to \vartheta$ *tal que* $\tau(E_i) = \omega_{\text{cent}}(A_i, E_i)$.
- $B = \{B_1, B_2, \ldots\}$*, tal que,* $B_i = (\vec{b}_{i1}, \ldots, \vec{b}_{i\mu})$*, donde* $\vec{b}_{ij} \in \mathcal{P}$ *(conjunto de fenotipos). La función de representación* $\Gamma : \mathcal{P} \to \mathcal{G}$ *es tal que* $\vec{b}_{ij} = \Gamma^{-1}(\vec{a}_{ij})$*. Podemos extender la notación para trabajar a alto nivel sobre toda la población:* $\Gamma^{-1}(A_i) = \left(\Gamma^{-1}(\vec{a}_{i1}, \ldots, \Gamma^{-1}(\vec{a}_{i\mu})\right)$.
- $\vartheta_B = \{\omega_{B\text{cent}}\}$ *donde* $\omega_{B\text{cent}}(B_i, E_i) = \Gamma^{-1} \circ \left(\omega_{\text{cent}}\left(\Gamma^{-1} \circ B_i, E_i\right)\right)$.
- $\tau_B : E \to \vartheta_B$ *se define como* $\tau_B(E) = \omega_{\text{cent}}(B_i, E)$.
- $E = \mathbb{R}^\mu$*, contiene los* valores de adecuación *de las estructuras en la población.* ■

El funcionamiento de un GA consiste pues en la repetida aplicación del operador de ciclo reproductor sobre una población de estructuras codificadas (A_i). Son los detalles de los operadores internos de selección (ω_s) y reemplazo (ω_r) los que diferencian los GAs generacionales ($\lambda = \mu$) de los GAs de estado estacionario ($\lambda = 1$ ó 2) –véase el Capítulo 2–. Observe el uso de conceptos más genéricos como conjunto de poblaciones posibles y entorno, ambos inspirados en la filosofía inicial de Holland [150].

Por otro lado, un algoritmo genético descentralizado (o de población estructurada) es un AS granulado de μ gránulos que se define como sigue:

Definición 4.4 (Algoritmo Genético Descentralizado) *Un* GA descentralizado *es una tupla*

$$\Xi_{\text{descent}} = (\Delta_{\text{descent}}, \Pi_{\text{descent}}, \xi_{\text{descent}}) \;\; , \tag{4.4}$$

en la que:

- $\Delta_{\text{descent}} = \{\Delta_1, \ldots \Delta_\mu\}$.
- $\Pi_{\text{descent}} \equiv \Delta_1 \| \Delta_2 \| \ldots \| \Delta_\mu$*, es decir, la política de activación es* concurrente *(implementada con paralelismo físico normalmente), usualmente sincronizada entre los gránulos.*
- $\xi_{\text{descent}}(\Delta_i, \Delta_j) = \begin{cases} \emptyset & j \notin \upsilon(i) \\ \chi_{ij}(B^i) & j \in \upsilon(i) \end{cases}$.

Esta última expresión caracteriza la comunicación entre dos gránulos. En ella, B^i *representa el conjunto de estructuras en el gránulo* Δ_i*. La función* χ_{ij} *selecciona de entre éstas a las estructuras distinguidas que se comparten con el gránulo* j*. Usualmente* $\chi_{ij} = \chi$*, es decir, el mecanismo de comunicación es homogéneo. La función* $\upsilon(i) : (N) \to P(\mathbb{N})$ *se denomina* función de vecindad *y determina qué gránulos comparten estructuras.* ■

En esta sección, hemos caracterizado los modelos secuenciales y de paralelización global mediante la definición de un sistema adaptativo granulado centralizado. Los dos modelos de algoritmos genéticos paralelos tradicionales, celular y distribuido, quedan caracterizados por la anterior definición de GA descentralizado. La diferencia entre ambos estriba en que, en los primeros, cada gránulo Δ_i contiene un vecindario propio $\upsilon(i) = \{\upsilon_1, \upsilon_2, \ldots, \upsilon_\mu^i\}$, siendo $\mu^i = |\upsilon(i)|$, y la función χ_{ij} selecciona siempre al mismo elemento distinguido de este vecindario ($\chi_{ij} \neq \chi_{ik}(b_i) \mid k \neq j;\ k, j \in \upsilon(i)$). En los segundos, cada gránulo contiene una subpoblación independiente (isla) de tamaño arbitrario, consistiendo usualmente la función χ_{ij} en la selección de la mejor estructura de cada isla o de una estructura aleatoria de ésta. En adición, los cGAs se sincronizan al final de cada ciclo reproductor (en el caso de los síncronos) o tras ejecutar el operador de reemplazo ω_r (en el caso de los asíncronos), mientras que los GAs distribuidos lo hacen con menor frecuencia. La función de comunicación ξ_{descent} forma parte del mecanismo de comunicación entre gránulos (individuos en el caso del cGA o islas para los dGAs).

4.2. Propuesta de Nuevos Modelos Eficientes

Es notable en la actualidad el interés por encontrar técnicas de optimización cada vez más eficientes sin que ello suponga la pérdida de capacidad de aplicación a nuevos problemas. En nuestro trabajo nos centramos en los algoritmos genéticos (GAs), que son técnicas que han demostrado ser muy eficientes en la resolución de problemas altamente complejos. Dentro de los GAs, se ha probado que estructurar la población nos lleva a obtener un mejor rendimiento con respecto al algoritmo equivalente en panmixia [27, 51]. En este trabajo estamos particularmente interesados en la familia de algoritmos genéticos celulares, un tipo de GA muy eficiente que ha sido explorado por la comunidad científica en mucha menor medida que en el caso de los GAs distribuidos.

Nuestra propuesta de trabajo se centra en la búsqueda de nuevos modelos de cGA para mejorar el comportamiento de un cGA canónico equivalente, y en la adaptación del modelo celular a otros dominios a los que no han sido exportados hasta ahora, con el objetivo de mejorar las propuestas existentes. Hay múltiples formas de lograr este objetivo. Por ejemplo, en la literatura reciente de cGAs podemos encontrar algunos ejemplos de trabajos que tratan de mejorar el rendimiento de estos algoritmos investigando en nuevos operadores genéticos [254], en el uso de distintas topologías de población [118, 227], en el ajuste del balance entre exploración/explotación [28, 214], etcétera.

En el caso que nos ocupa, nuestro esfuerzo se ha centrado (ver Figura 4.1) en el diseño de innovadores algoritmos que mejoren el equilibrio entre la exploración y la explotación que aplican sobre el espacio de búsqueda (cGAs jerárquicos, adaptativos, algoritmos meméticos celulares), en la adaptación de algunos de los modelos desarrollados a nuevos dominios a los que no se habían aplicado los cGAs, y en la búsqueda de modelos paralelos eficientes.

Creemos que una metodología muy apropiada para mejorar el comportamiento de una heurística es trabajar sobre sus principios de funcionamiento. Para ello, es necesario estudiar y caracterizar el algoritmo teóricamente, aunque muchos investigadores no consideren este importante paso. Nosotros, en cambio, sí realizamos este estudio en nuestras investigaciones (ver Sección 4.1 y el Capítulo 5). Quizás la característica que influye más notablemente en el comportamiento de un algoritmo de optimización sea la relación entre exploración y explotación que éste aplica sobre el espacio de búsqueda. Éstas, son dos características contrapuestas (es decir, que están en conflicto entre sí) que deben convivir en el algoritmo para alcanzar el éxito. Como ya se ha comentado con anterioridad, una heurística con una elevada capacidad de explotación convergerá muy pronto hacia un óptimo local, del que seguramente no podrá escapar. Por el contrario, si el algoritmo potencia demasiado la exploración del espacio de

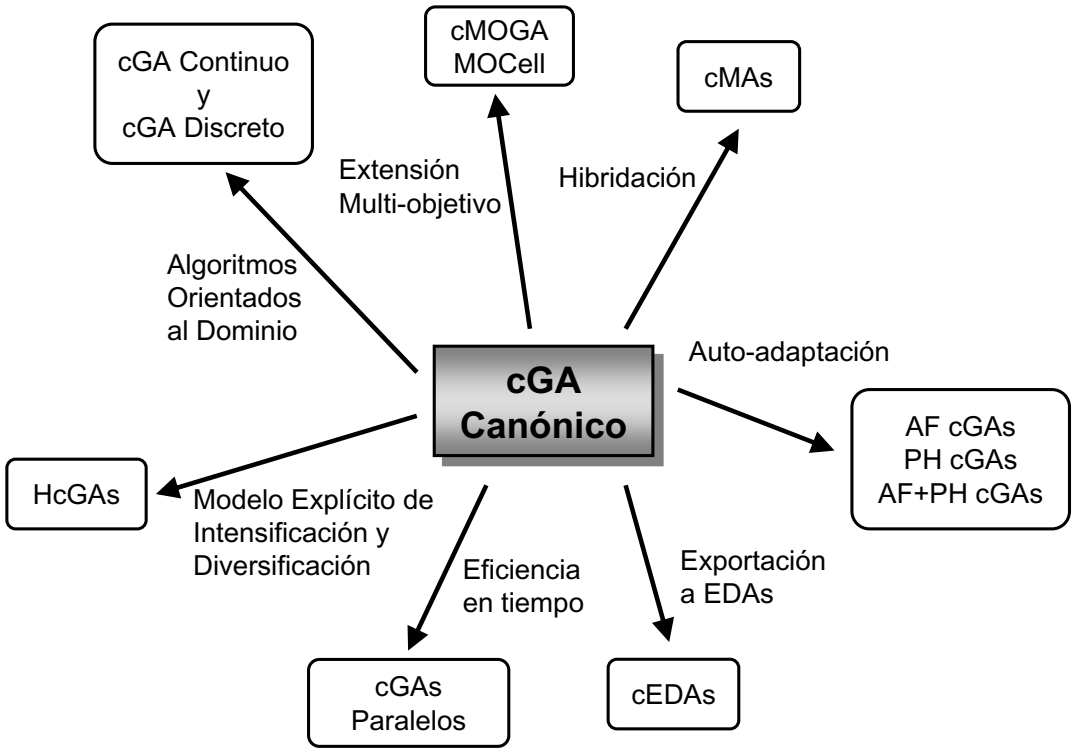


Figura 4.1: Extensiones y mejoras que planteamos hacer sobre un cGA canónico en nuestra propuesta de solución al problema considerado en esta tesis.

búsqueda, será capaz de recorrer una parte importante del espacio de soluciones del problema, pero no podrá profundizar en ninguna región, por lo que, probablemente, encontrará soluciones de calidad pobre. Un equilibrio adecuado entre exploración y explotación le permitirá al algoritmo recorrer un amplio espectro del espacio de búsqueda (exploración), profundizando en las zonas más prometedoras (explotación). Así mismo, es deseable que el algoritmo mantenga en todo momento su capacidad de exploración ya que será la responsable de permitir a la heurística escapar de óptimos locales.

Por todo esto es por lo que en este trabajo ponemos un especial esfuerzo en el estudio teórico del comportamiento de los cGAs, y por lo que proponemos algunos nuevos modelos que tratan de mejorar el equilibrio entre exploración y explotación de estos algoritmos. De esta forma, pretendemos desarrollar mejoras algorítmicas al modelo canónico de cGA que nos permitan obtener un rendimiento superior al de éstos. Las extensiones propuestas siguen distintas filosofías generales para conseguir algoritmos eficientes y precisos al mismo tiempo. Además, dichas mejoras algorítmicas son con frecuencia acumulativas, lo que significa por ejemplo que podemos combinar las ventajas de dos de nuestras propuestas en otros nuevos algoritmos en el futuro (por ejemplo, cEDAs meméticos, cEDAs adaptativos o cGAs paralelos multi-objetivo). Los modelos algorítmicos que proponemos se resumen a continuación:

- **cGAs adaptativos.** Proponemos en el Capítulo 6 tres nuevos algoritmos que ajustan automáticamente el equilibrio entre exploración y explotación en función de la velocidad de convergencia de la población. De esta forma, si la población converge demasiado rápidamente, se potenciará la exploración del algoritmo para ralentizar la pérdida de diversidad, mientras que si la convergencia es lenta, se pondrá mayor énfasis en la explotación local de las soluciones. Se han utilizado tres métodos diferentes para medir la velocidad de convergencia de la población, dando lugar a los tres cGAs propuestos: AF cGA, en el que la velocidad de convergencia se mide según la media de los valores de fitness de los individuos de la población, PH cGA, en cuyo caso se utiliza la entropía de la población, y AF+PH cGA, que utiliza los dos valores, fitness medio y entropía.

- **cGAs Jerárquicos.** Se presenta en el Capítulo 7 un nuevo modelo de cGA para mejorar el equilibrio entre exploración y explotación del cGA canónico. En este caso, se establece una jerarquía en la población colocando a las soluciones en una disposición de niveles piramidal, de forma que los mejores individuos se encuentran en el centro de la población (nivel más bajo), y cuanto más nos alejemos del centro (ascendemos de nivel) peores serán los individuos. Con esta jerarquía logramos potenciar la explotación de las mejores soluciones, mientras que en los niveles altos de la jerarquía mantenemos la diversidad de soluciones.

- **EDAs celulares.** Para el estudio realizado en el Capítulo 8 nos planteamos que puede ser interesante exportar las ventajas que tiene el modelo celular a otras familias de EAs distintas de los GAs. En este caso, hemos propuesto un nuevo modelo algorítmico de EDA en el que la población se encuentra estructurada en una malla toroidal. Se trata del primer EDA celular.

- **Algoritmos meméticos celulares (cMAS).** Los algoritmos meméticos son algoritmos altamente especializados. En ellos se utiliza información del problema a resolver en la representación de los individuos o los operadores de variación, por ejemplo. Además, suelen incorporar un método de búsqueda local que ayude a refinar las soluciones. En el Capítulo 9 presentamos el primer cMA en la literatura, según nuestro conocimiento. El algoritmo desarrollado ha sido aplicado al problema SAT (una descripción del problema está disponible en el Apéndice A). Además, proponemos el estudio de otro cMA para el caso del problema de rutas de vehículos en el Capítulo 13.

- **cGAs paralelos.** En el Capítulo 10 proponemos dos nuevos modelos paralelos de cGAs. Uno de ellos está diseñado para ejecutarse en redes de área local y es fiel al modelo de cGA canónico, mientas que el otro es un modelo distribuido en islas, dentro de las cuales se ejecutan cGAs. Este segundo modelo está diseñado para ejecutarse en entornos de *grid computing*.

- **cGAs Multi-objetivo.** También se han realizado dos propuestas de cGAs al dominio de la optimización multi-objetivo: MOCell (Capítulo 11) y cMOGA (Capítulo 15). MOCell es una evolución de cMOGA que logra obtener mejores resultados, pero que (aún) no está preparado para trabajar con individuos heterogéneos (con genes de distintos tipos), como se requiere en el problema abordado en el Capítulo 15.

- **cGAs en dominios continuos.** Por último, gracias a las características de los GAs, podemos aplicar nuestro algoritmo a algunos otros dominios de optimización simplemente definiendo un nuevo tipo de individuo para tal dominio. En el Capítulo 14 hemos sacado provecho de esta propiedad aplicando JCell al dominio de optimización continua.

4.3. Evaluación de los Resultados

Como ya se ha comentado en varias ocasiones a lo largo de este documento, el trabajo realizado durante esta tesis consiste en la propuesta y evaluación de nuevas técnicas y mejoras algorítmicas sobre cGAs. Como es sabido, los GAs pertenecen al campo de las metaheurísticas debido a que son técnicas no deterministas. Esto implica que dos ejecuciones del mismo algoritmo sobre un problema dado no tienen por qué encontrar la misma solución. Esta propiedad característica de las metaheurísticas supone un problema importante para los investigadores a la hora de evaluar sus resultados y, por tanto, a la hora de comparar su algoritmo con otros algoritmos existentes.

Existen algunos trabajos que abordan el análisis teórico para un gran número de heurísticas y problemas [138, 159]; pero dada la dificultad que entraña este tipo de análisis teórico, tradicionalmente se analiza el comportamiento de los algoritmos mediante comparaciones empíricas. Para ello, las técnicas estudiadas se aplican a un amplio rango de problemas bien conocidos de diversas características. De esta manera, los algoritmos se pueden comparar en función de la calidad de las soluciones obtenidas y del esfuerzo computacional empleado en encontrarlas. Este esfuerzo lo calculamos en términos del número de evaluaciones de la función de fitness realizado durante la búsqueda.

A lo largo de los estudios realizados en esta tesis doctoral, siempre se ha tratado de seguir una misma metodología. Esta metodología puede dividirse en tres etapas o pasos principales:

1. El primer punto consiste en la definición de los objetivos que se pretenden alcanzar, de las instancias de problemas sobre las que vamos a trabajar, y de los factores en los que nos centraremos para evaluar y analizar nuestros resultados.

2. Una vez hecho esto, el segundo paso será definir las métricas que utilizaremos para medir los resultados, ejecutar los experimentos para obtener dichos resultados, y analizar los datos obtenidos. En las secciones 4.3.1 y 4.3.2 se presentan las métricas que hemos utilizado en nuestro trabajo, así como la metodología empleada para analizar los resultados.

3. Por último, consideramos que es muy importante dar información de cómo se han realizado los experimentos, con el fin de que puedan ser reproducidos por otros investigadores. Para ello, se debe facilitar toda la información relativa a los algoritmos utilizados (código, parámetros, etc.), a los problemas estudiados (instancias, tamaño, características, referencias, etc.), e incluso al entorno de ejecución (memoria, procesador, sistema operativo, lenguaje de programación, etc.). La forma en la que se presenta en un trabajo toda la información es de gran importancia. Esta información se debe mostrar de manera que resulte fácilmente asimilable por el lector sin penalizar la claridad de los contenidos, para así evitar cualquier tipo de confusión por parte del lector.

4.3.1. El Caso Mono-Objetivo

Debido a la naturaleza no determinista de los algoritmos estudiados, las comparaciones sobre los resultados de una única ejecución no son consistentes, por lo que deben ser realizadas sobre un conjunto grande de resultados, obtenidos tras realizar un elevado número de ejecuciones independientes del algoritmo sobre el problema dado. Para poder realizar las comparaciones sobre los resultados obtenidos debemos recurrir al uso de funciones estadísticas. En la literatura, tradicionalmente se han empleado funciones estadísticas muy sencillas, tales como la media de los valores obtenidos, la mediana, o la mejor y peor soluciones encontradas por la técnica estudiada. Pero estos no son valores concluyentes, y pueden

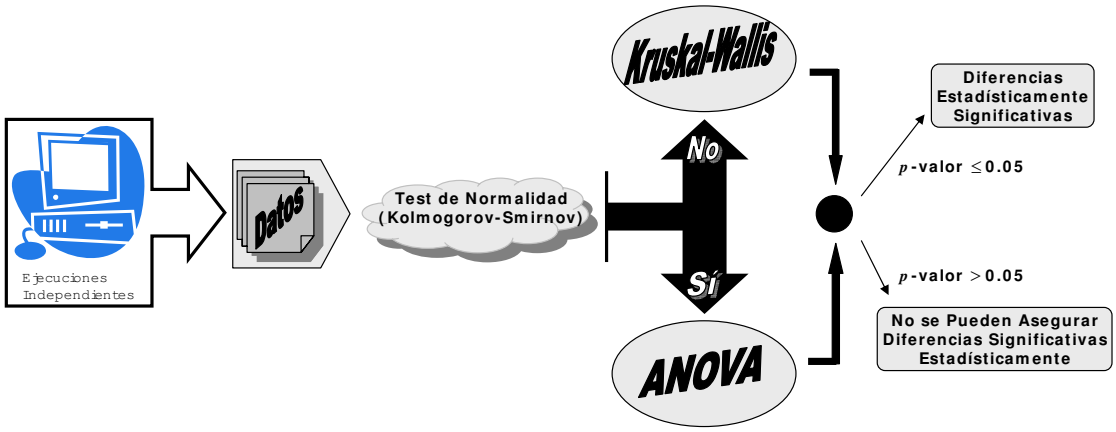


Figura 4.2: Esquema del proceso de evaluación de resultados.

llevar a interpretaciones erróneas de los resultados. Por tanto, debemos recurrir a estudios estadísticos que nos permitan asegurar que los resultados y comparaciones presentados en nuestros estudios son significativos.

Con el fin de obtener resultados con *significancia estadística*, los investigadores suelen recurrir al uso de t-tests o análisis de varianza (ANOVA). El uso de estas funciones estadísticas nos permite determinar si los efectos observados en los resultados obtenidos son significativos o si, por el contrario, son debidos a errores en el muestreo realizado. Pero el uso de la función ANOVA sobre cualquier distribución de datos no es del todo apropiado, puesto que en el caso de una distribución no normal de datos puede ser erróneo el resultado. En este caso, sería más correcto aplicar un test de Kruskal-Wallis.

En la Figura 4.2 presentamos la metodología seguida para la evaluación e interpretación de los resultados obtenidos en los experimentos realizados a lo largo de todo el trabajo desarrollado para esta tesis doctoral. Para obtener los resultados de nuestras pruebas se han realizado un mínimo de 30 ejecuciones independientes del algoritmo sobre el problema dado, aunque en la mayoría de los estudios aquí presentados, el número de ejecuciones independientes realizados asciende a 100. Una vez obtenidos los resultados de las ejecuciones, se les aplica un test de Kolmogorov-Smirnov con el fin de comprobar si los valores de los resultados siguen una distribución normal (gaussiana) o no. Si la distribución es normal, se realiza un test de ANOVA, mientras que si no lo es, el test que aplicamos es el de Kruskal-Wallis. En este trabajo consideramos en todo momento un nivel de confianza del 95 % (nivel de significancia del 5 % o p-valor por debajo de 0.05) en nuestros tests estadísticos. Esto significa que podemos asegurar que las diferencias son significativas (no son fruto de la casualidad) con una probabilidad del 95 %.

En algunos de los estudios desarrollados en este trabajo de tesis se utilizan problemas multi-objetivo, que son problemas en los que se deben optimizar más de una función simultáneamente (los algoritmos utilizados para este tipo de problema se denominan algoritmos multi-objetivo). Además, las funciones que componen un problema multi-objetivo suelen estar en conflicto, lo que significa que mejorar uno de los objetivos implica empeorar alguno de los demás. Por tanto, las soluciones a este tipo de problemas no son un único valor, sino un conjunto de puntos no dominados (para más detalles consulte el Capítulo 11). Es necesario, por tanto, recurrir al uso de métricas para poder comparar distintos resultados de este tipo de problemas. La aplicación de estas métricas nos permite comparar los algoritmos, e incluso la aplicación de los análisis estadísticos comentados más arriba para poder obtener confianza estadística en nuestras conclusiones. En la Sección 4.3.2 se presentan las métricas empleadas en nuestros estudios.

4.3.2. El Caso Multi-Objetivo

Como se ha introducido previamente, en el caso de la optimización multi-objetivo necesitamos utilizar métricas para poder comparar la calidad de las soluciones de distintos algoritmos. Una vez aplicadas estas métricas, podemos proceder a realizar sobre los resultados obtenidos las mismas pruebas estadísticas propuestas en el caso mono-objetivo. Actualmente, no existe ninguna métrica en la literatura que nos permita asegurar la superioridad de un algoritmo multi-objetivo frente a otro para un conjunto de problemas dado. Por tanto, se deben utilizar varias métricas que se centren en diversos aspectos de las soluciones para comparar los algoritmos estudiados. Antes de aplicar algunas de las métricas, es conveniente normalizar los valores sobre los que se aplican para obtener resultados fiables. Tal y como se propone en [78] aplicamos la normalización con respecto al frente óptimo, si se conoce, o con respecto a la mejor y peor solución de nuestro frente en el caso de que no se conociera el óptimo. En los trabajos realizados para esta tesis se han empleado las siguientes métricas:

- **Distancia generacional – GD.** Esta métrica fue introducida por Van Veldhuizen y Lamont [282] para medir cómo de lejos están los elementos del conjunto de vectores encontrados no dominados del conjunto óptimo de Pareto; se define como:

$$GD = \frac{\sqrt{\sum_{i=1}^{n} d_i^2}}{n} \ , \tag{4.5}$$

  donde n es el número de vectores en el conjunto de soluciones no dominadas, y d_i es la distancia Euclídea (medida en el espacio objetivo) entre cada una estas soluciones y el miembro más cercano del conjunto óptimo de Pareto. Según esta definición, es claro que un valor de $GD = 0$ significa que todos los elementos generados están en el conjunto óptimo de Pareto.

- **Dispersión – Δ.** La métrica de *dispersión* [80] es una medida de diversidad que cuantifica la magnitud que alcanza la dispersión entre las soluciones obtenidas. Esta métrica se define como:

$$\Delta = \frac{d_f + d_l + \sum_{i=1}^{N-1} \left| d_i - \bar{d} \right|}{d_f + d_l + (N-1)\bar{d}} \ , \tag{4.6}$$

  donde d_i es la distancia Euclídea entre dos soluciones consecutivas, $\bar{d}$ es la media de estas distancias, y d_f y d_l son las distancias Euclídeas a las soluciones *extremas* (límite) del frente exacto de Pareto en el espacio objetivo (para más detalles, consulte [80]). Esta medida toma el valor cero para una distribución ideal, remarcando una dispersión perfecta de las soluciones del frente de Pareto.

- **Hipervolumen – HV.** La métrica *hipervolumen* [302], que es una métrica combinada de convergencia y diversidad, calcula el volumen (en el espacio de objetivos) cubierto por los miembros de un conjunto Q de soluciones no-dominadas (la región acotada por la línea discontinua en la Figura 4.3, $Q = \{A, B, C\}$) para problemas en los que todos los objetivos deben ser minimizados. Matemáticamente, para cada solución $i \in Q$, un hipercubo v_i se construye utilizando un punto de referencia W (que puede estar compuesto por la peor solución para cada objetivo, por ejemplo) y la solución i como las esquinas de la diagonal del hipercubo. El punto de referencia se puede encontrar simplemente construyendo un vector de los peores valores para las funciones. El hipervolumen se calcula como el volumen de la unión de todos los hipercubos:

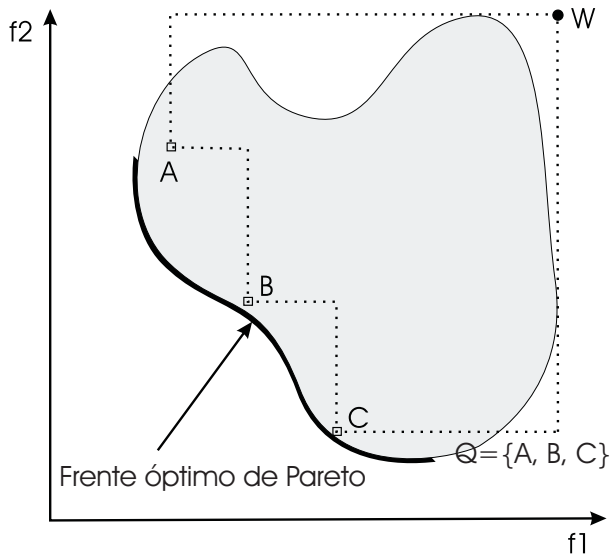


Figura 4.3: El hipervolumen cercado por las soluciones no dominadas.

$$HV = volumen\left(\bigcup_{i=1}^{|Q|} v_i\right) \quad . \tag{4.7}$$

- **Número de óptimos de Pareto.** En la resolución de algunos problemas multi-objetivo muy complejos, encontrar un número alto de soluciones no dominadas puede ser una tarea realmente dura para el algoritmo. En este sentido, el número de óptimos de Pareto encontrados se puede utilizar como una medida de la capacidad del algoritmo para explorar los espacios de búsqueda definidos por el problema multi-objetivo.

- **Cubrimiento de conjuntos – C(A, B).** La métrica del *cubrimiento de conjuntos* $C(A, B)$, calcula la proporción de soluciones en B que son dominadas por soluciones de A:

$$C(A, B) = \frac{|\{b \in B | \exists a \in A : a \preceq b\}|}{|B|} \quad . \tag{4.8}$$

  Un valor de la métrica $C(A, B) = 1$ significa que todos los miembros de B son dominados por A, mientras que $C(A, B) = 0$ significa que ningún miembro de B es dominado por A. De esta forma, cuanto mayor sea $C(A, B)$, mejor es el frente de Pareto A con respecto a B. Ya que el operador de dominancia no es simétrico, $C(A, B)$ no es necesariamente igual a $1 - C(B, A)$, y tanto $C(A, B)$ como $C(B, A)$ deben ser calculados para entender cuántas soluciones de A son cubiertas por B y viceversa.

4.3.3. Algunas Definiciones Adicionales

Presentamos en esta sección algunas definiciones que se han utilizado en nuestros estudios bien para obtener algunos de los valores presentados en las tablas y figuras o bien en alguno de los modelos algorítmicos. En concreto, damos las definiciones del fitness medio y de la entropía de una población de individuos, y de funciones matemáticas como la desviación estándar, la mediana y el rango intercuartílico.

Definición 4.5 (Fitness medio $\overline{f}$) *El fitness medio de los individuos de una población P de tamaño N se define como la media de la suma de los valores de fitness de todos los individuos que forman la población:*

$$\overline{f} = \frac{\sum_{i=1}^{N} f_i}{N} \ , \tag{4.9}$$

donde f_i es el valor de fitness del individuo i. ■

Definición 4.6 (Entropía de la población H_p) *La entropía de una población H_p se calcula como la media de las entropías obtenidas para cada gen de los cromosomas de los individuos:*

$$H_p = \frac{\sum_{i=1}^{L} H_i}{L} \ , \tag{4.10}$$

donde H_i es la entropía del conjunto de valores formados por el gen en la posición i del cromosoma de cada individuo i, que tiene longitud L. ■

Antes de definir la desviación estándar, necesitamos conocer el concepto de varianza, que se da a continuación:

Definición 4.7 (Varianza S) *Definimos la varianza o la desviación media cuadrática de una muestra de n observaciones $X_1, X_2, \ldots X_n$ como:*

$$S = \frac{\sum_{i=1}^{n} (X_i - \overline{X})^2}{n-1} \ , \tag{4.11}$$

donde $\overline{X}$ es la media aritmética de la muestra. ■

La desviación estándar puede, pues, definirse como:

Definición 4.8 (Desviación estándar *stdv*) *La desviación estándar se define como la raíz cuadrada de la varianza:*

$$stdv = \sqrt{\frac{\sum_{i=1}^{n} (X_i - \overline{X})^2}{n-1}} \ . \tag{4.12}$$

■

Cuando se presenta alguna observación extrema, la media se ve afectada. En este caso, resulta apropiado utilizar la mediana, en lugar de la media, para describir el conjunto de datos:

Definición 4.9 (mediana $\widetilde{x}$) *La* mediana $\widetilde{x}$ *de un conjunto ordenado de valores* $\{x_1, \ldots, x_n\}$ *es el valor que se encuentra en el centro de una secuencia ordenada* x_i*, siendo* $i = (n+1)/2$. ■

Finalmente, la definición del rango intercuartílico es:

Definición 4.10 (rango intercuartílico IQR) *El rango intercuartílico* IQR *es una medida de la dispersión de un conjunto de valores. Se calcula simplemente como la diferencia entre el tercer cuartil y el primero:*

$$IQR = Q_3 - Q_1 \ . \tag{4.13}$$

■

4.4. Conclusiones

En este capítulo presentamos una caracterización generalizada de los GAs descentralizados. Tras esto, describimos las nuevas formas teóricas de mejora de las que son susceptibles los cGAs. De entre ellas, describimos las propuestas de las que se derivan los modelos estudiados en los capítulos siguientes.

Por otro lado, mostramos en la segunda parte del capítulo la metodología que hemos seguido para analizar los resultados obtenidos, así como para comparar nuestros algoritmos con otros previamente existentes. La razón de la descripción de esta metodología se basa en que un mal planteamiento en la evaluación de los resultados puede llevar a resultados inconsistentes o erróneos.

Capítulo 5

Nueva Propuesta Teórica: Estudios sobre la Presión de Selección en cGAs

Para introducir el necesario contenido teórico de base en esta tesis caracterizamos en este capítulo la presión que ejerce la selección del algoritmo sobre la población, una propiedad básica que influye en su comportamiento de manera muy importante. Como hemos mencionado, cualquier algoritmo requiere un adecuado equilibrio entre exploración y explotación para resultar útil en la práctica. En algoritmos basados en población la presión selectiva es un aspecto clave que representa este equilibrio en gran manera.

Las variaciones en la forma de la topología y la definición de vecindad pueden dar lugar a infinidad de algoritmos distintos, pero es posible un estudio metodológico mediante la definición de un radio para la forma de la población y el vecindario (definido en la Sección 2.5.2). La relación entre ambos, llamada ratio (ver Sección 2.5.2), define las características de la búsqueda, ya que tiene influencia sobre el equilibrio entre exploración y explotación que aplica el algoritmo sobre la población de soluciones.

Existen muy pocos trabajos que traten de caracterizar el comportamiento de los cGAs. Algunos de ellos proponen ecuaciones matemáticas que modelan el comportamiento de la presión de selección del algoritmo, pero no existe en la actualidad ningún modelo matemático que se ajuste bien a la curva de la presión de selección de cGAs con poblaciones rectangulares. Nosotros en este trabajo hacemos este estudio tanto en el campo teórico como en el práctico.

En este capítulo definiremos los conceptos de presión de selección y el *tiempo de takeover* (en la Sección 5.1). Tras esto, realizaremos en la Sección 5.2 un estudio teórico del comportamiento de los cGAs en función de su ratio, presentaremos y analizaremos los modelos matemáticos existentes en la literatura para aproximar el comportamiento de los cGAs, y proporcionaremos dos nuevos modelos matemáticos que permiten predecir la presión de selección de un cGA en función del valor de su ratio. Finalmente, contrastaremos en la Sección 5.3 los resultados teóricos obtenidos mediante el estudio del comportamiento de varios cGAs con distintas ratios sobre un conjunto de siete tipos de problemas de complejidad considerable (pertenecientes a los campos de la optimización numérica y combinatoria) que sustentarán nuestras conclusiones. Veremos que es difícil decidir cuál es la mejor ratio según la familia de problemas.

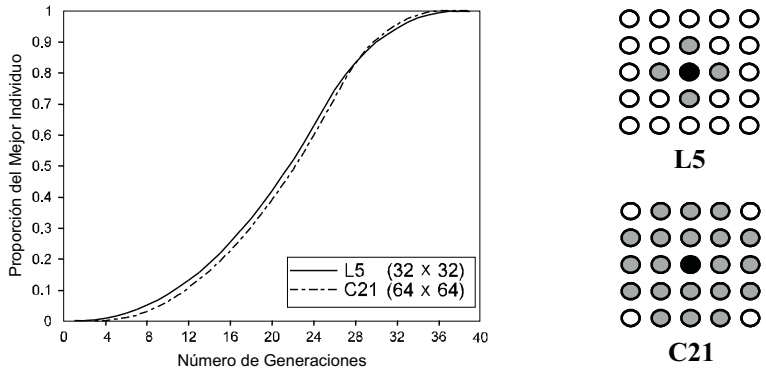


Figura 5.1: Curvas de crecimiento del mejor individuo para dos cGAs con formas de vecindario y población diferentes, pero teniendo valores de ratio similares.

5.1. La Presión de Selección

La *presión de selección* está relacionada con el concepto del *tiempo de takeover*, que se define como el tiempo necesario para que un único (mejor) individuo colonice la población completa con copias de él mismo utilizándose únicamente el operador de selección [122, 243]. Tiempos de *takeover* menores son indicativos de una intensidad mayor en la selección aplicada por el algoritmo sobre la población (una mayor presión de selección) y, por tanto, una mayor velocidad de convergencia del algoritmo.

Los algoritmos que tienen un valor de la *ratio* parecido muestran una presión de selección similar, como se concluye en [247]. En la Figura 5.1 mostramos el comportamiento similar de dos algoritmos celulares con diferentes radios de vecindario y población, pero con valores de la ratio similares. Los algoritmos de la gráfica usan un vecindario NEWS con una población de 32×32 individuos, y un vecindario C21 con una población de 64×64 individuos. El eje de ordenadas representa la proporción de la población formada por copias del mejor individuo en función del tiempo.

Por tanto, resulta muy interesante estudiar cómo influyen algunos parámetros del cGA en el comportamiento de búsqueda del algoritmo. En concreto, la política de actualización de los individuos de la población es un parámetro influyente en la presión de selección del algoritmo que ha sido estudiado teóricamente en trabajos previos a esta tesis [116, 119]. Así, si mantenemos la forma del vecindario y de la malla constantes (por ejemplo, L5 y una población cuadrada) pero utilizamos diversas políticas de actualización de los individuos, podemos apreciar distintos comportamientos del algoritmo. De hecho, se puede observar en la Figura 5.2 que la presión de selección global inducida por las diversas políticas asíncronas se encuentran en una zona intermedia entre los límites del caso síncrono y el de la elevada presión de selección del caso panmíctico. Por tanto, variando la política de actualización de los individuos podemos influir en las capacidades de exploración o explotación de la búsqueda realizada por el algoritmo. La gráfica mostrada en dicha figura ha sido obtenida tras hacer la media de 100 ejecuciones independientes, utilizando una selección por torneo binario sobre una población de 32×32 individuos con un vecindario L5. La selección por torneo binario consiste en elegir aleatoriamente dos individuos del vecindario y escoger el de mayor calidad.

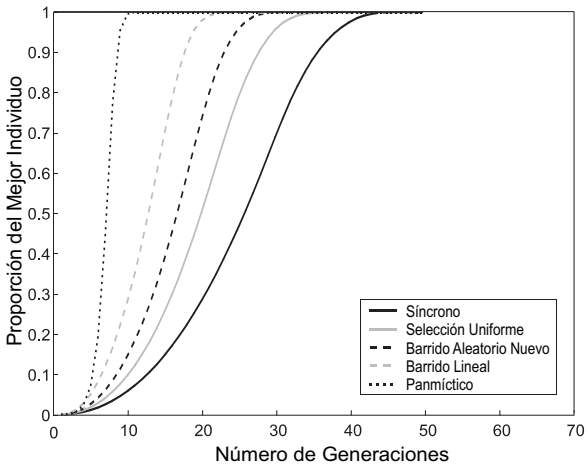


Figura 5.2: Presión de selección de varios cGAs idénticos, pero con distintas políticas de actualización. Selección por torneo binario.

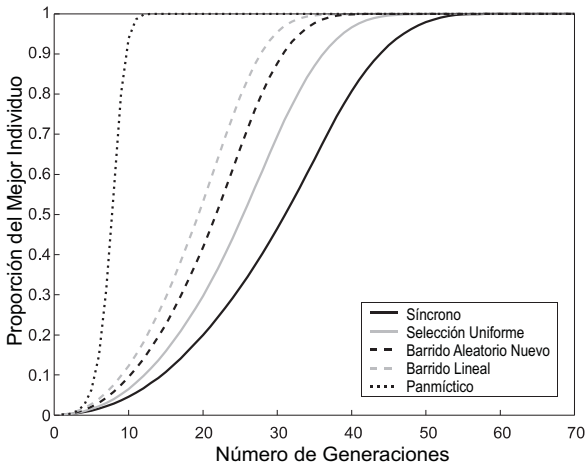


Figura 5.3: Presión de selección de varios cGAs idénticos, pero con distintas políticas de actualización. Selección por ruleta.

Otro parámetro que influye en la presión de selección de un cGA es el operador de selección de los padres. Los cGAs de las gráficas de la presión de selección mostradas en la Figura 5.2 utilizan selección por torneo binario. Se pueden apreciar las diferencias con las curvas mostradas en la Figura 5.3, que se corresponden con la presión de selección de exactamente los mismos algoritmos, pero con la salvedad de que en este caso se usa el operador de selección por ruleta. Puede comprobarse observando ambas figuras que el uso del operador de la ruleta induce una menor presión de selección (tiempos de *takeover* mayores) en el algoritmo que el torneo binario.

En esta tesis, proponemos también el estudio de la influencia de la ratio entre los radios de vecindario y población sobre la presión de selección de un cGA [20, 87]. Para ello, calculamos la gráfica de la presión de selección para varios cGAs que difieren entre ellos únicamente en la forma de la población utilizada, mientras que el tamaño permanece constante. Debemos remarcar que, dado que no se incluyen los operadores, los resultados para los cGAs presentados tienen además una amplia extensión a otros cGAs, por lo que en muchas ocasiones los mencionamos para recordar este interesante hecho. En la Figura 5.4 se muestra la presión de selección de diferentes cGAs síncronos utilizando el vecindario L5 y las 6 posibles formas de malla para una población compuesta de 1024 individuos. Al igual que en el caso de los cGAs asíncronos, los datos han sido obtenidos tras calcular la media de 100 ejecuciones independientes de cada algoritmo. Nótese que la presión de selección inducida sobre mallas rectangulares en cGAs síncronos se encuentra por debajo de la curva del caso del cGA síncrono utilizando la malla cuadrada (población de 32×32 individuos), lo que significa que las mallas más estrechas favorecen un estilo de búsqueda más explorativo. En el caso de la Figura 5.4, debe tenerse en cuenta que las diferencias en el eje de las abscisas están dispuestas en escala logarítmica.

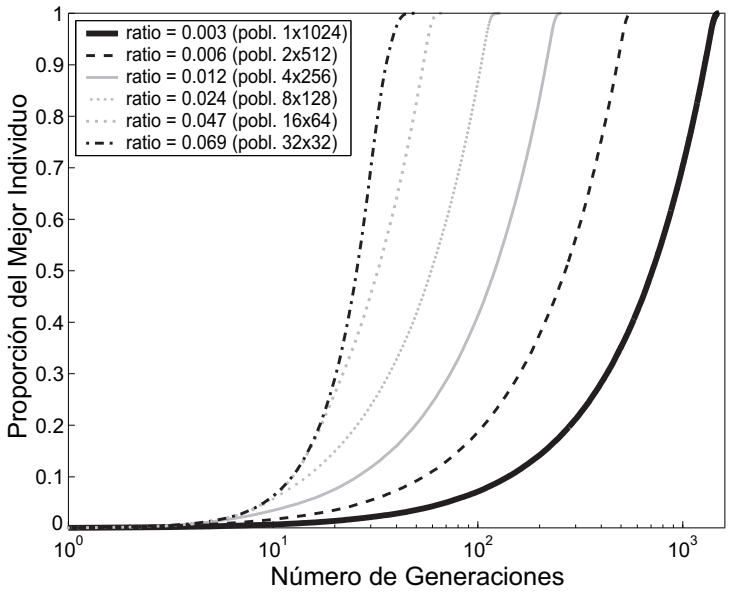


Figura 5.4: Presión de selección de varios cGAs idénticos pero con distinta forma de la población.

5.2. Estudio Teórico

Además del tiempo de *takeover*, otro aspecto importante para analizar el comportamiento de un EA son las curvas de crecimiento. En [24] se caracteriza el comportamiento de EAs distribuidos en términos del tiempo de *takeover* y de las curvas de crecimiento en función del porcentaje y la frecuencia de migración de individuos entre las islas.

Como se comentó en el Capítulo 3, se han realizado varios estudios tratando de caracterizar matemáticamente el comportamiento de los GAs celulares, pero sólo uno de ellos contempla la posibilidad del uso de poblaciones rectangulares [119]. Como se puede ver en la Figura 5.4, la ratio entre el vecindario y la población influye marcadamente en el comportamiento del cGA. Si mantenemos constante el vecindario utilizado, cambiar la forma de la población implica el cambio en el valor de la ratio. En esta sección, estudiaremos los modelos existentes para aproximar el comportamiento de un cGA, tras lo cual propondremos dos nuevos modelos (polinómico y probabilístico) para caracterizar matemáticamente la curva de crecimiento de un cGA en función del valor de la ratio. Para ello, mantenemos constante durante todo nuestro estudio la forma del vecindario (empleamos el vecindario NEWS), y caracterizamos el comportamiento del cGA en función de la forma de la población utilizada.

5.2.1. Aproximación al Modelo Determinista

Si consideramos un comportamiento determinista en la presión de selección de un cGA el mejor vecino de cada individuo será seleccionado siempre, por lo que si un individuo tiene un vecino de fitness alto, pasará a formar parte del conjunto de mejores individuos en la siguiente generación. Como consecuencia, sólo se expanden los individuos de fitness alto por las celdas que se encuentran en el límite de la región de mejores individuos, y lo hacen con una probabilidad del 100 %, tal y como se muestra en la Figura 5.5.

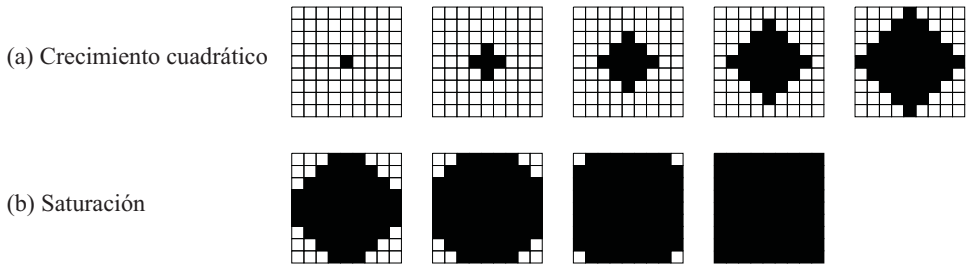


Figura 5.5: Crecimiento determinista de un cEA con población cuadrada.

Por tanto, la proporción del mejor individuo crece de forma cuadrática hasta que se alcanzan los límites de la malla de la población (última imagen de la Figura 5.5a). Esto sucede en el tiempo t', cuando $N(t') = K/2$, es decir, cuando la mitad de las celdas de la población contienen una copia del mejor individuo (K es el tamaño de la población). Tras esto, la tasa de crecimiento disminuye de la misma forma en que antes se incrementaba, como se muestra en la Figura 5.6.

$$N(t) = \begin{cases} a\,t^2 & \text{si } t < T/2, \\ -a\,(t-T)^2 + 1 & \text{en otro caso.} \end{cases} \tag{5.1}$$

Podemos dar una definición de la curva de crecimiento con la Ecuación 5.1, donde se asume por simplicidad que $N(0) = 0$. La tasa de crecimiento viene dada por el parámetro de ajuste a. Durante la primera mitad del proceso de *takeover*, $t < T/2$ (siendo T el tiempo total de *takeover*), el crecimiento es cuadrático. En el caso de la segunda mitad la curva decrece en la misma proporción en la que creció con anterioridad.

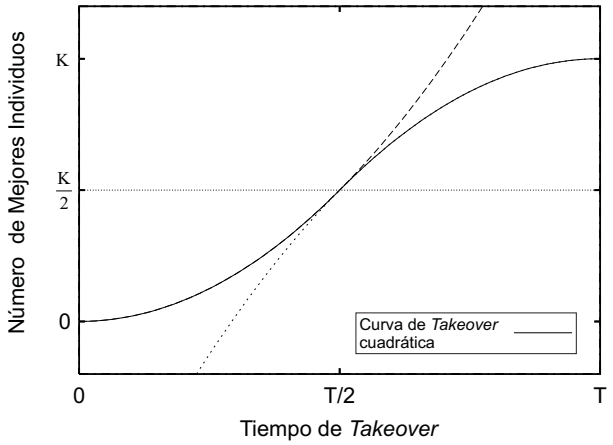


Figura 5.6: Suponiendo un comportamiento determinista de la presión de selección, la proporción del mejor individuo crece de forma cuadrática durante la mitad del proceso, y después desciende a la misma velocidad con la que creció.

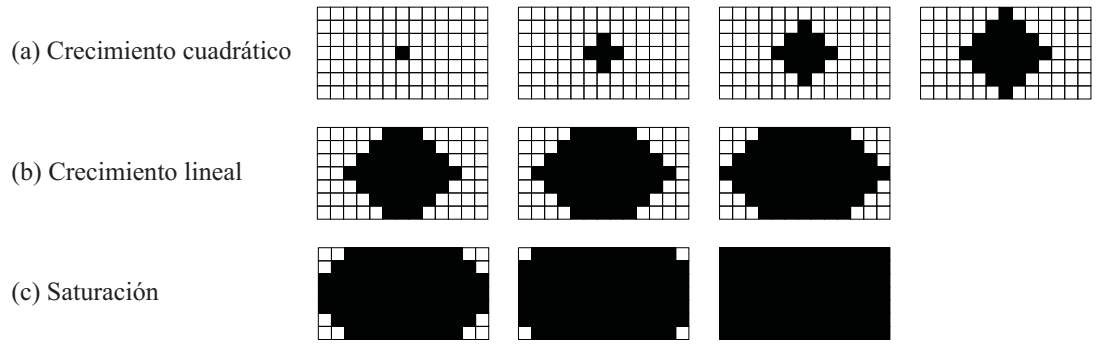


Figura 5.7: Crecimiento determinista de un cEA con población rectangular.

En el caso de utilizar una población rectangular, la curva de crecimiento cambia. Como se puede ver en la Figura 5.7, la proporción del mejor individuo en la población sigue, como en el caso de la malla cuadrada, un crecimiento cuadrático hasta que se alcanzan los bordes de la población. Para la malla rectangular, es evidente que no se pueden alcanzar simultáneamente los cuatro bordes de la población, sino que se alcanzarán en primer lugar los bordes de la población por la parte estrecha. El crecimiento de la proporción del mejor individuo de la población desde que se alcanzan los dos primeros bordes de la población (por la parte estrecha) hasta que se alcanzan los otros (por la parte ancha) es lineal. Tras esto, la curva decrece en la misma proporción en que se incrementaba en la primera parte del crecimiento, como sucedía en el caso de la malla cuadrada. En la Figura 5.8 se muestra una gráfica de la presión de selección de un cGA con una población rectangular cuando el crecimiento es determinista. En ella, puede apreciarse claramente el crecimiento lineal que se produce entre la curva de crecimiento cuadrático inicial (Figura 5.7a) y la curva final de saturación (Figura 5.7c).

Por tanto, el crecimiento del número de mejores individuos en la población es cuadrático hasta el momento $t = M/2$ (en que el mejor individuo se extiende hasta los bordes de la población por la parte estrecha, ver Figura 5.7a), siendo $M \leq N$, a partir de este momento el número de mejores individuos crece de forma lineal hasta que se alcanzan los otros bordes de la población (por la parte ancha, como se ve en la Figura 5.7b), en el tiempo $t = N/2$. Finalmente, en una tercera etapa (Figura 5.7c), la curva decrece con la misma velocidad en que crecía en la primera etapa. En la Ecuación 5.2 se define matemáticamente la curva de crecimiento teórica mostrada en la Figura 5.8.

$$N(t) = \begin{cases} a\,t^2 & \text{si } t < M/2 \ , \\ b\,t & \text{si } t < N/2 \ , \\ -a\,(T-t)^2 + 1 & \text{en otro caso} \ . \end{cases} \tag{5.2}$$

5.2.2. Modelos Existentes de Aproximación al Comportamiento Real de cGAs

En esta sección estudiamos diversas aproximaciones a la curva de la presión de selección de los cGAs con distintas formas de población bidimensionales. Para ello, utilizamos los principales modelos existentes: el modelo logístico propuesto por Sarma y De Jong en [246], el modelo basado en hipergrafos propuesto por Sprave en [260], y el modelo propuesto por Giacobini et al. [119].

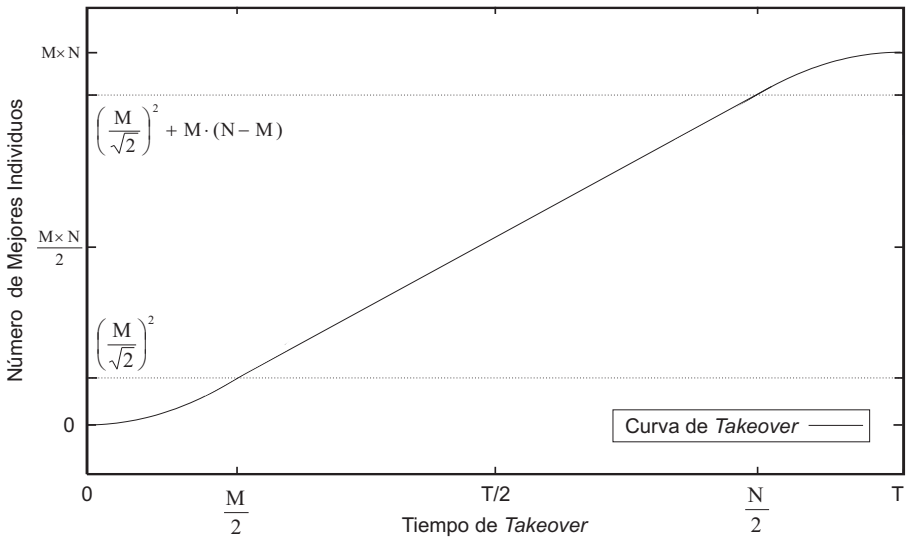


Figura 5.8: Crecimiento del mejor individuo en un cGA con población M×N (M≤N), suponiendo un comportamiento determinista en la presión de selección.

Como se puede ver en la Tabla 5.1, el cGA que pretendemos aproximar en esta sección está compuesto por una población de alrededor de 4096 individuos, y se han estudiado 25 formas de mallas toroidales distintas para albergar los individuos. Uno de los padres es siempre el individuo actual, mientras que el otro es elegido por torneo binario. Como se ha dicho anteriormente, en el estudio de la presión de selección de un algoritmo no se emplean operadores de recombinación ni mutación, y el individuo es sustituido por el mejor de los padres si es mejor que el individuo actual.

El error ha sido calculado como la media para todas las distintas poblaciones del error cuadrático medio del modelo aproximado con respecto a los datos experimentales. Es decir, sean los vectores $\vec{a}$ y $\vec{b}$ los valores obtenidos por nuestra aproximación y por el cGA, respectivamente, el error Δ entre nuestra aproximación y el cGA para las 25 poblaciones estudiadas se define como:

$$\Delta = \frac{\sum_{i=1}^{25} \mathrm{ECM}(\mathrm{modelo}_i)}{25} \;\; ; \;\; \mathrm{ECM}(\mathrm{modelo}) = \frac{1}{l}\sum_{i=0}^{l}(\vec{a}[i] - \vec{b}[i])^2 \;\; , \tag{5.3}$$

siendo l la longitud de los vectores $\vec{a}$ y $\vec{b}$.

Tabla 5.1: Parametrización utilizada en los cGAs.

Tamaño de la Población	≈4096 Individuos
Selección de los Padres	Individuo Actual + Torneo Binario
Recombinación	ninguna
Mutación	ninguna
Reemplazo	Reempl_si_Mejor
Fitness Mejores Individuos	1.0
Fitness Peores Individuos	0.0

Aproximación Logística

En esta sección vamos a mostrar el comportamiento de la función logística propuesta por Sarma y De Jong en [246] para aproximar la curva de la presión de selección de los cGAs. La función propuesta en ese trabajo –$LOG(t)$ en la Ecuación 5.4– resultó ser una buena aproximación para cGAs con poblaciones cuadradas, pero en cambio no funciona bien en el caso de utilizar poblaciones rectangulares, como se puede ver en la Figura 5.9. Con el fin de mejorar la aproximación de la función logística para las poblaciones rectangulares, se ha probado en este estudio añadir un parámetro más a dicha función, obteniendo $LOG2(t)$ como resultado.

$$LOG(t) = \frac{1}{1 + \left(\frac{1}{P(0)} - 1\right) \cdot e^{-at}} \;\; ; \;\; LOG2(t) = \frac{1}{1 + (b-1) \cdot e^{-at}} \;\; . \tag{5.4}$$

En la Figura 5.9 se muestran las curvas obtenidas con los dos modelos logísticos junto con la del cGA. Como se puede ver, $LOG2$ (con error $\Delta = 9.5446 \cdot 10^{-4}$) se aproxima mucho mejor a la curva del cGA que LOG ($\Delta = 2.4303 \cdot 10^{-2}$). De cualquier forma, ninguna de las dos funciones logísticas se aproxima con exactitud a la curva del cGA, sobre todo en el caso de las poblaciones rectangulares (ratios 0.003 y 0.0158).

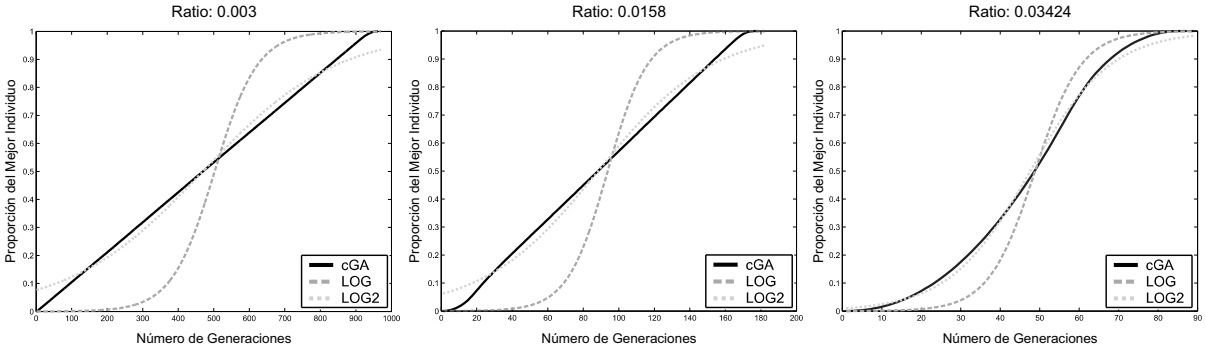


Figura 5.9: Aproximación de los dos modelos logísticos a las curvas de convergencia de tres cGAs.

El Modelo de Hipergrafos

Sprave desarrolló en [260] un método para estimar la curva de crecimiento de la presión de selección basado en hipergrafos. Para utilizar este método, es necesario calcular del diámetro de la estructura de la población y la probabilidad de la distribución inducida por el operador de selección. De hecho, Chakraborty et al. calcularon previamente [54] las probabilidades de éxito (p_{select}) de los operadores de selección más conocidos, lo que representa un aspecto importante para ser utilizado en el modelo de hipergrafos.

Se han utilizado en este estudio dos definiciones diferentes de la probabilidad de éxito del operador de selección por torneo binario –ver la Ecuación 5.5–. Una de ellas (p_{select}) es la propuesta en [54], donde N representa el tamaño de la población e i es el número de mejores individuos en la población. Por otro lado, p_{select2} es una adaptación que se propone en este trabajo para mejorar la aproximación del modelo de hipergrafos, donde s es la probabilidad de que el individuo dado sea de fitness elevado, y es un valor que no es necesario calcular específicamente para esta función, sino que se calcula al aproximar la presión de selección siguiendo el modelo de hipergrafos, que se puede consultar en [260].

$$p_{\text{select}} = \frac{i}{N} + \left(1 - \frac{i}{N}\right) \cdot \left(2 \cdot \frac{i}{N} - \left(\frac{i}{N}\right)^2\right) \;\; ; \;\; p_{\text{select2}} = s + (1-s) \cdot \left(2 \cdot \frac{i-s}{N} - \left(\frac{i-s}{N}\right)^2\right) \;\; . \quad (5.5)$$

En la Figura 5.10 mostramos la aproximación de los dos modelos de hipergrafos propuestos, HG1 (en el que usamos p_{select}) y HG2 (que utiliza la modificación propuesta p_{select2}), al comportamiento del cGA en tres poblaciones distintas. Como se puede ver, el modelo HG2 (con $\Delta = 1.9859 \cdot 10^{-2}$) se aproxima mejor a la curva del cGA que HG1 ($\Delta = 4.2969 \cdot 10^{-2}$). En cualquier caso, las aproximaciones que realizan estos dos modelos distan aún del comportamiento real del cGA, al igual que sucedió con las funciones logísticas. De todas formas, observamos que las curvas obtenidas con HG1 y HG2 tienen formas similares a las del cGA, aunque no se aproximen con exactitud.

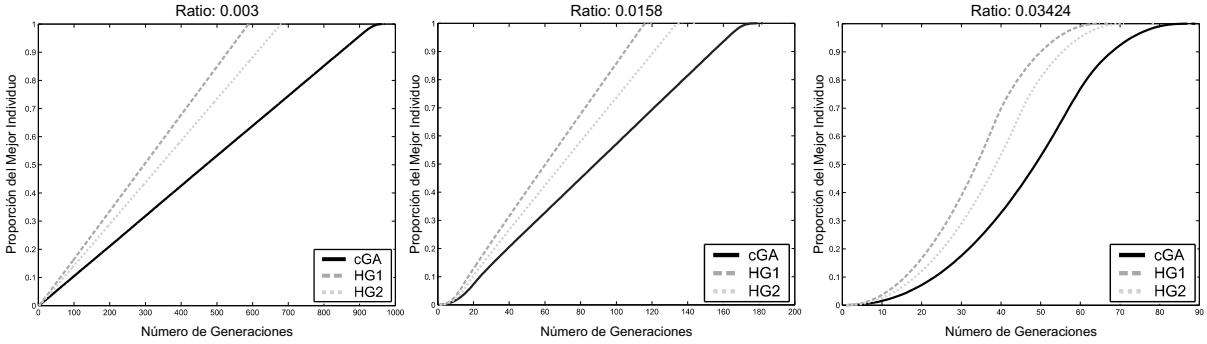


Figura 5.10: Aproximación de los modelos basados en hipergrafos a las curvas de convergencia de tres cGAs.

Recurrencia Probabilística de Giacobini et al.

De entre los modelos propuestos en [119], destacamos en esta sección el caso del cGA síncrono con una población toroidal, que definen los autores con la siguiente ecuación de recurrencia:

$$\begin{cases} T(0) = 1 & \\ T(t) = T(t-1) + 4 \cdot p_2 \frac{\sqrt{T(t-1)}}{\sqrt{2}} & \text{si } T(t) \leq M^2/2 \\ T(t) = T(t-1) + 2 \cdot (M-1) \cdot p_2 + p_1 & \text{si } T(t) \leq N \cdot M - M^2/4 \\ T(t) = T(t-1) + 4 \cdot p_2 \sqrt{n - T(t-1)} & \text{en otro caso} \end{cases} , \quad (5.6)$$

donde la dimensión de la población es $n = M \times N$, con $M \leq N$, y p_1 y p_2 son las probabilidades de éxito (de que uno de los mejores individuos del vecindario sea seleccionado) si hay 1 ó 2 individuos de fitness alto en el vecindario, respectivamente. Para el caso que nos ocupa, en el que queremos modelar el comportamiento de un cGA con selección por torneo binario, los valores que toman estas probabilidades son $p_1 = 9/25$ y $p_2 = 16/25$.

En la Figura 5.11 se puede apreciar que la aproximación propuesta por Giacobini et al. en [119] es, con diferencia, la que mejor se aproxima de los modelos existentes en la literatura, obteniendo un error medio $\Delta = 7.6735 \cdot 10^{-4}$.

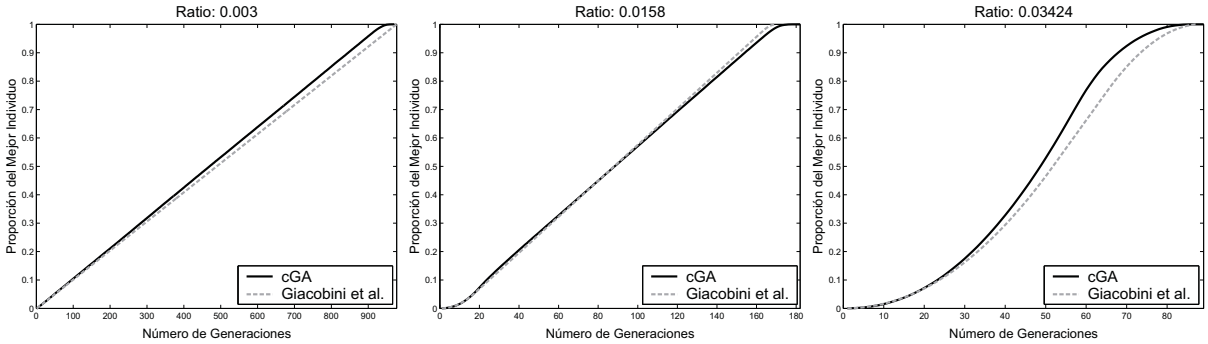


Figura 5.11: Aproximación de la recurrencia probabilística de Giacobini et al. a las curvas de convergencia de tres cGAs.

5.2.3. Dos Nuevos Modelos de Aproximación

En esta sección investigamos dos nuevos modelos matemáticos que nos permitan predecir la curva de la presión de selección de un cGA con una mayor precisión que los existentes en la literatura, presentados en la Sección 5.2.2. Los dos nuevos modelos propuestos están basados en un polinomio de grado 3 y un modelo probabilístico.

Aproximación Polinómica

Tras los estudios realizados en la Sección 5.2.2, parece claro que la presión de selección de los cGAs estudiados con poblaciones rectangulares no sigue un modelo logístico. Por tanto, nos proponemos en esta sección aproximar el comportamiento del cGA utilizando un polinomio de grado 3 con cuatro parámetros de la forma: $POL3(t) = at^3 + bt^2 + ct + d$. Como se muestra en la Figura 5.12, $POL3$ se aproxima al comportamiento del cGA mejor que cualquiera de los modelos previamente existentes, ya que el error medio cometido por $POL3$ es $\Delta = 9.8627 \cdot 10^{-5}$, un orden de magnitud menor que el mejor de los modelos existentes previamente: el propuesto por Giacobini et al., con $\Delta = 7.6735 \cdot 10^{-4}$.

Sin embargo, $POL3$ cuenta con el inconveniente de estar en función de cuatro parámetros, cuyos valores son independientes de las características del algoritmo a aproximar, a diferencia del modelo propuesto por Giacobini et al., que está definido en función de la forma de la población y de la probabilidad de éxito del operador de selección. Por ello, se trata en la Ecuación 5.7 de dar un polinomio de grado 3 que esté en función del tamaño de la población y de la ratio entre los tamaños de población y vecindario, y que se aproxime lo mejor posible a la curva de la presión de selección del cGA (n es el tamaño de la población). El ajuste de este polinomio a la curva del cGA es algo peor que en el caso de $POL3$ (error $\Delta = 2.1609 \cdot 10^{-4}$ de POL frente a $\Delta = 9.8627 \cdot 10^{-5}$ en el caso de $POL3$), pero sigue siendo mejor que el obtenido con la función de Giacobini et al. Sin embargo, este modelo polinómico tiene el problema de ser poco intuitivo y, además, no contempla el uso de otros operadores de selección.

$$POL(t) = -\frac{3 \cdot n}{2}\text{ratio}^6 \cdot t^3 + \frac{15 \cdot n}{2}\text{ratio}^5 \cdot t^2 + (-4 \cdot n \cdot \text{ratio}^4 + 0.35 \cdot \text{ratio}) \cdot t + \text{ratio} \ . \tag{5.7}$$

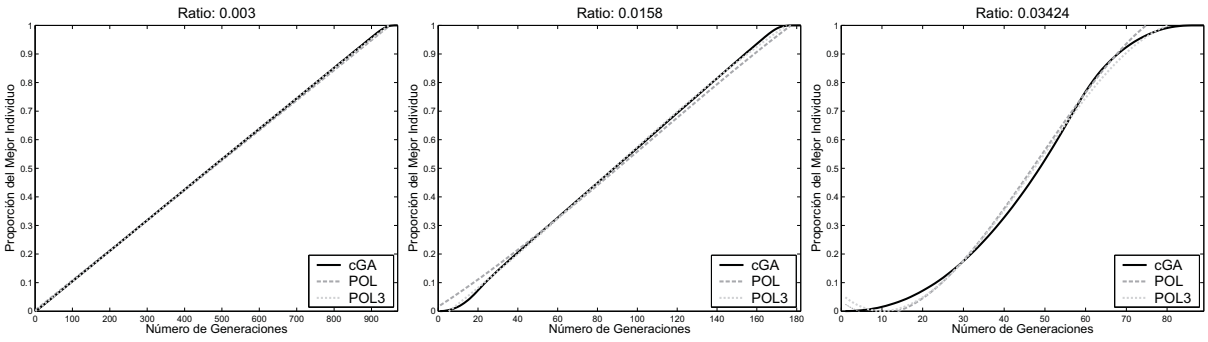


Figura 5.12: Aproximación de los dos modelos polinómicos a las curvas de convergencia de tres cGAs.

Modelo Probabilístico

Nos proponemos en esta sección obtener un modelo matemático no parametrizado y que sea capaz de aproximarse con el menor error posible a la curva de la presión de selección de cGAs reales con diferentes operadores de selección y distintas formas de la malla. El modelo matemático que proponemos está basado en probabilidades, como se puede ver en la Ecuación 5.8.

Como se propuso en la Sección 5.2.1, dividimos la curva de la presión de selección de un cGA en tres tramos. Esto nos permite aproximar nuestra función para el caso de las poblaciones con formas tanto cuadradas como rectangulares. En el caso de un modelo real, debe tenerse en cuenta que el uso de diferentes métodos de selección implica una distinta expansión del número de mejores individuos por la población. Como consecuencia, la velocidad de expansión de las mejores soluciones por la población es menor que en el caso determinista, y depende del método de selección utilizado. Por lo tanto, los límites de transición entre las diferentes funciones que forman el modelo que fueron fijados en el caso determinista deben ponderarse por un factor dependiente del método de selección empleado. Se le ha llamado a este factor v, y es un parámetro que representa la velocidad con la que se expanden los mejores individuos por la población. Este parámetro se define como el inverso de la probabilidad de que un individuo de fitness bajo situado en la frontera con el conjunto de mejores individuos pase a formar parte de este conjunto en la siguiente generación. En la Ecuación 5.8 se presenta el nuevo modelo propuesto, donde por simplicidad no se ha especificado que $N(0) = 1$.

La primera función trata de aproximar la curva desde $t = 0$ hasta el momento $t = v \cdot M/2$ (suponiendo que $M \leq N$), momento en el que los bordes de la expansión de los mejores individuos de la población se conectan (por el camino más corto). Desde este punto, y hasta el momento en el que el conjunto de mejores individuos se conecte por el camino más largo $t = v \cdot N/2$, el número de individuos de fitness bajo que pasarán a ser de fitness alto en cada generación sigue un crecimiento lineal. Finalmente, el tercer tramo de nuestra aproximación es una función de saturación que termina cuando no quede en la población ningún individuo de fitness bajo.

$$T(t) = \begin{cases} T(t-1) + 4 \cdot (t-1) \cdot (p_2 \cdot p_2^2 + 2 \cdot (1-p_2) \cdot p_2 \cdot p_1) + 4 \cdot p_2 \cdot p_1 & t < \frac{M}{2} \cdot v \ , \\ T(t-1) + 2 \cdot M \cdot p_2 + 2 \cdot p_2 \cdot p_1 & \frac{M}{2} \cdot v \leq t < \frac{N}{2} \cdot v \ , \\ T(t-1) + 4 \cdot \sum_{j=0}^{\lfloor \frac{M-1}{2} \rfloor - 1} \left(\left\lfloor \frac{M-1}{2} \right\rfloor - j \right) \cdot (p_2 + 4 \cdot (1-p_2) \cdot p_2 \cdot p_1) & \text{en otro caso} \ . \end{cases} \tag{5.8}$$

En las siguientes secciones, definimos los valores de los parámetros empleados en la Ecuación 5.8 para 75 cGAs diferentes, con 25 formas distintas de población (con un tamaño de 4096 individuos), utilizando el vecindario NEWS. En estos algoritmos, se considera que uno de los padres es siempre el propio individuo, mientras que el otro padre puede ser seleccionado por los métodos de la ruleta, torneo u ordenación lineal. Además, calculamos el error de nuestra aproximación con respecto a los datos obtenidos por el algoritmo, y presentamos algunas figuras para mostrar gráficamente cómo se ajusta nuestra aproximación al comportamiento real del algoritmo.

Selección por Ruleta

La selección por ruleta consiste en seleccionar, de entre todos los individuos del vecindario, a cada padre con una cierta probabilidad, que está definida como la relación entre su valor de fitness y la suma total de los valores de fitness de todos los individuos de la vecindad. Matemáticamente, se muestra en la Ecuación 5.9 la probabilidad de que un individuo i sea seleccionado como padre cuando se utiliza la selección por ruleta.

$$p_i = \frac{\text{fitness}(i)}{\sum_{j \in \text{Vecindario}} \text{fitness}(j)} \tag{5.9}$$

En el caso del estudio de la presión de selección, sólo existen dos posibles valores de fitness que pueden tomar los individuos. Estos valores son 1.0 y 0.0. Por tanto, según la Ecuación 5.9, la probabilidad de que un individuo de fitness bajo (con fitness = 0.0) sea seleccionado por el método de la ruleta es 0.0, por lo que si existe algún vecino de fitness alto (con fitness = 1.0) en el vecindario del individuo considerado, éste (u otro con el mismo valor de fitness en el caso de que existieran más) será seleccionado como padre con toda seguridad. Por lo tanto, tanto p_1 como p_2 tendrán valor 1.0 en este modelo y, por consiguiente $v = 1.0$. Como consecuencia, ya que tanto p_1 como p_2 toman el valor 1.0 al utilizar la selección por ruleta, el algoritmo mostrará un comportamiento determinista.

En la Figura 5.13 se muestra la aproximación obtenida por la función probabilística propuesta a las curvas de la presión de selección de cuatro cGAs con distintas poblaciones, seleccionados de entre los 25 algoritmos estudiados. Como se puede apreciar, el ajuste obtenido es bastante preciso en los cuatro casos, siendo el error medio obtenido para todas las poblaciones $\Delta = 1.4904 \cdot 10^{-6}$, que es cuatro órdenes de magnitud menor que el error obtenido por la recurrencia probabilística de Giacobini et al. cuando se utiliza el operador de selección por ruleta: $\Delta = 0.0194$.

Selección por Torneo Binario

En la selección por torneo binario, dos individuos son elegidos aleatoriamente del vecindario, y el mejor de ellos es el considerado como padre. Por tanto, si existen en el vecindario dos individuos con fitness = 1.0, la probabilidad de elegir uno de ellos como padre es $p_2 = 16/25$, mientras que si sólo existiera un mejor individuo en el vecindario, la probabilidad de elegirlo sería $p_1 = 9/25$. El valor de v será en este caso 25/16.

Se muestran en la Figura 5.14 las curvas obtenidas por la aproximación probabilística y el cGA para las mismas cuatro poblaciones estudiadas en la Figura 5.13. Como se puede ver, aunque la aproximación no es tan buena como en el caso de la ruleta, consideramos que sigue siendo una importante mejora, ya que el error cometido es $\Delta = 2.5788 \cdot 10^{-4}$, menor que la mitad del error cometido en el caso del modelo propuesto por Giacobini et al. Debe tenerse en cuenta además que la aproximación en este caso es más

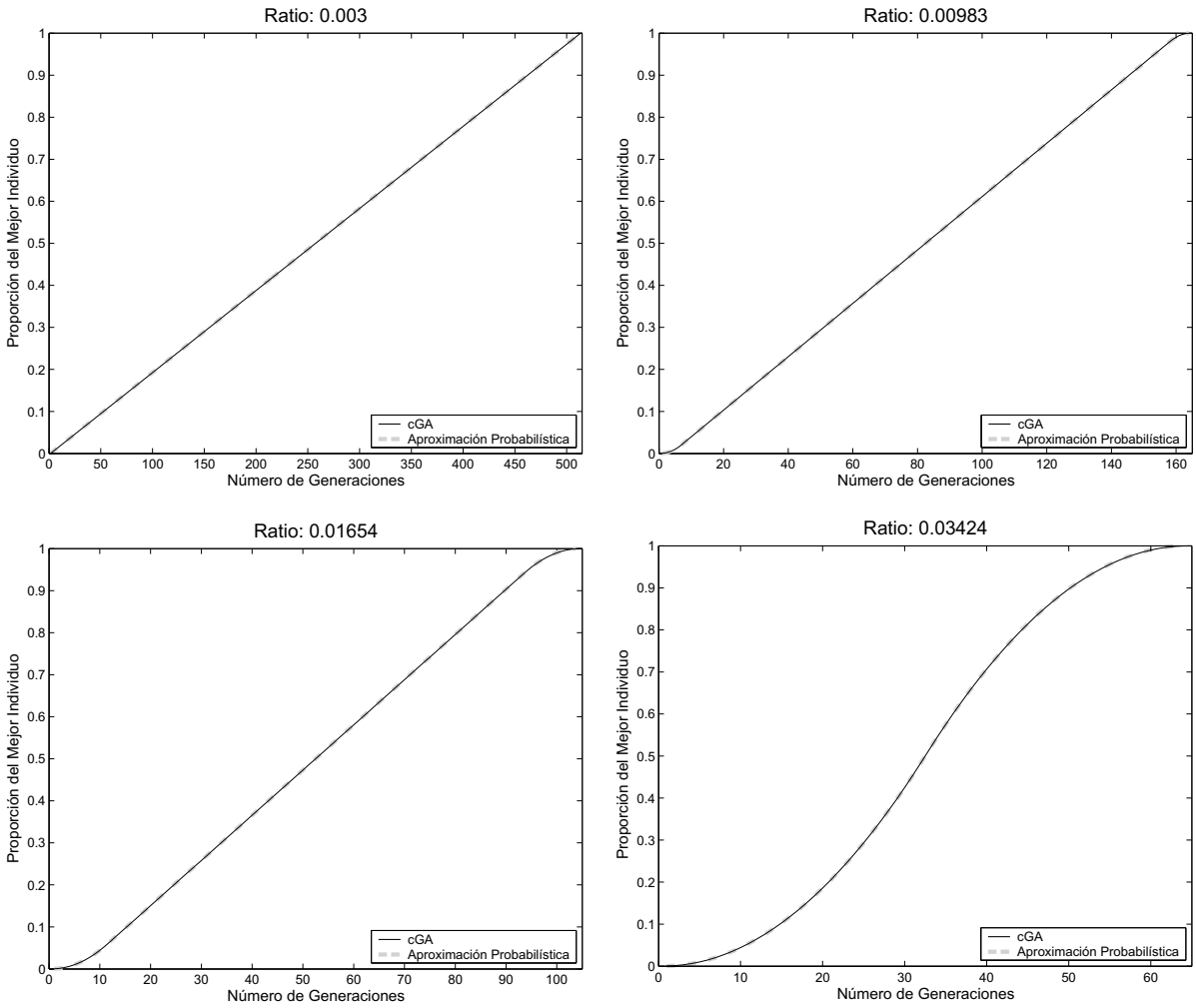


Figura 5.13: Aproximación probabilística a las curvas convergencia de cuatro cGAs. Selección por ruleta.

complicada que en el caso de utilizar la selección por ruleta, ya que el comportamiento del cGA no es determinista, y de ahí la pérdida de dos órdenes de magnitud obtenida en el error cometido.

Selección por Ordenación Lineal

En el caso de la selección por ordenación lineal, todos los individuos del vecindario son ordenados en una lista según su valor de fitness (de mejor a peor), y el padre será seleccionado con una probabilidad mayor cuanto mejor sea su clasificación en dicha lista. En concreto, la probabilidad de que el individuo i sea seleccionado para la fase de reemplazo es:

$$p_i = \frac{2 \cdot (s - j)}{s \cdot (s - 1)} \ , \tag{5.10}$$

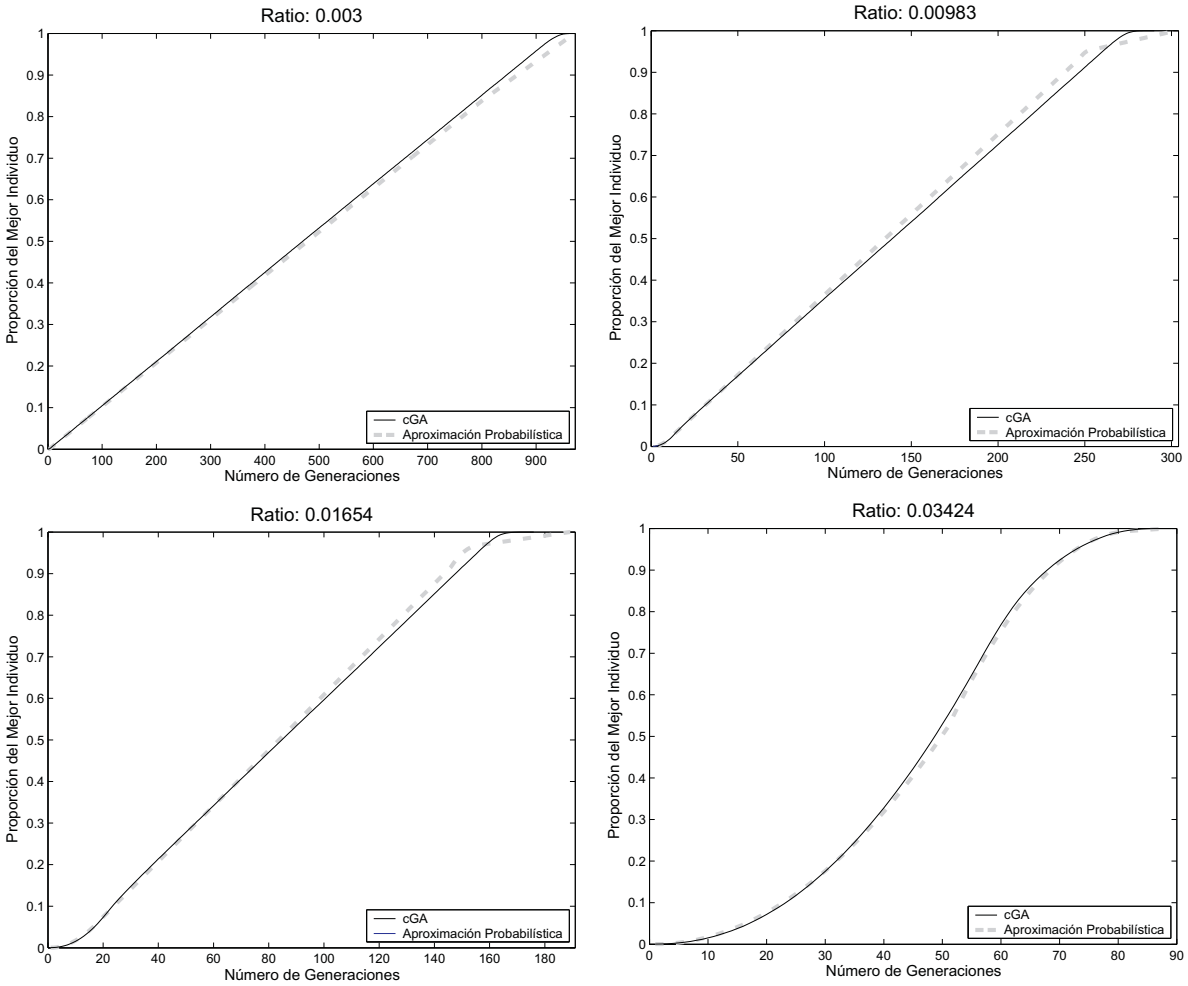


Figura 5.14: Aproximación probabilística a las curvas convergencia de cuatro cGAs. Selección por torneo binario.

donde s es el tamaño del vecindario (la longitud de la clasificación) y j es el puesto que tiene el individuo i en la clasificación.

Para el caso de la selección por ordenación lineal, definimos las probabilidades de escoger un individuo de fitness elevado del vecindario como $p_1 = 2/5$ y $p_2 = 7/10$ para los casos en los que tengamos uno o dos individuos de fitness alto en el vecindario, respectivamente.

Al igual que sucedió en el estudio del torneo binario, la aproximación no es en este caso tan buena como la que se obtuvo para la ruleta. La razón vuelve a ser la dificultad que supone modelar el comportamiento no determinista del cGA. En la Figura 5.15 mostramos las gráficas con la aproximación del modelo al cGA para las cuatro poblaciones estudiadas en las dos secciones anteriores. El error medio obtenido en este caso es $\Delta = 4.0705 \cdot 10^{-4}$, que se encuentra en el mismo orden de magnitud que el obtenido para el torneo binario, y un orden de magnitud mayor que en el caso de la ruleta. Comparado con el modelo propuesto por Giacobini et al. aplicado al caso de la selección por ordenación lineal, que obtiene un error $\Delta = 0.0017$, el modelo probabilístico propuesto en esta tesis es dos órdenes de magnitud mejor.

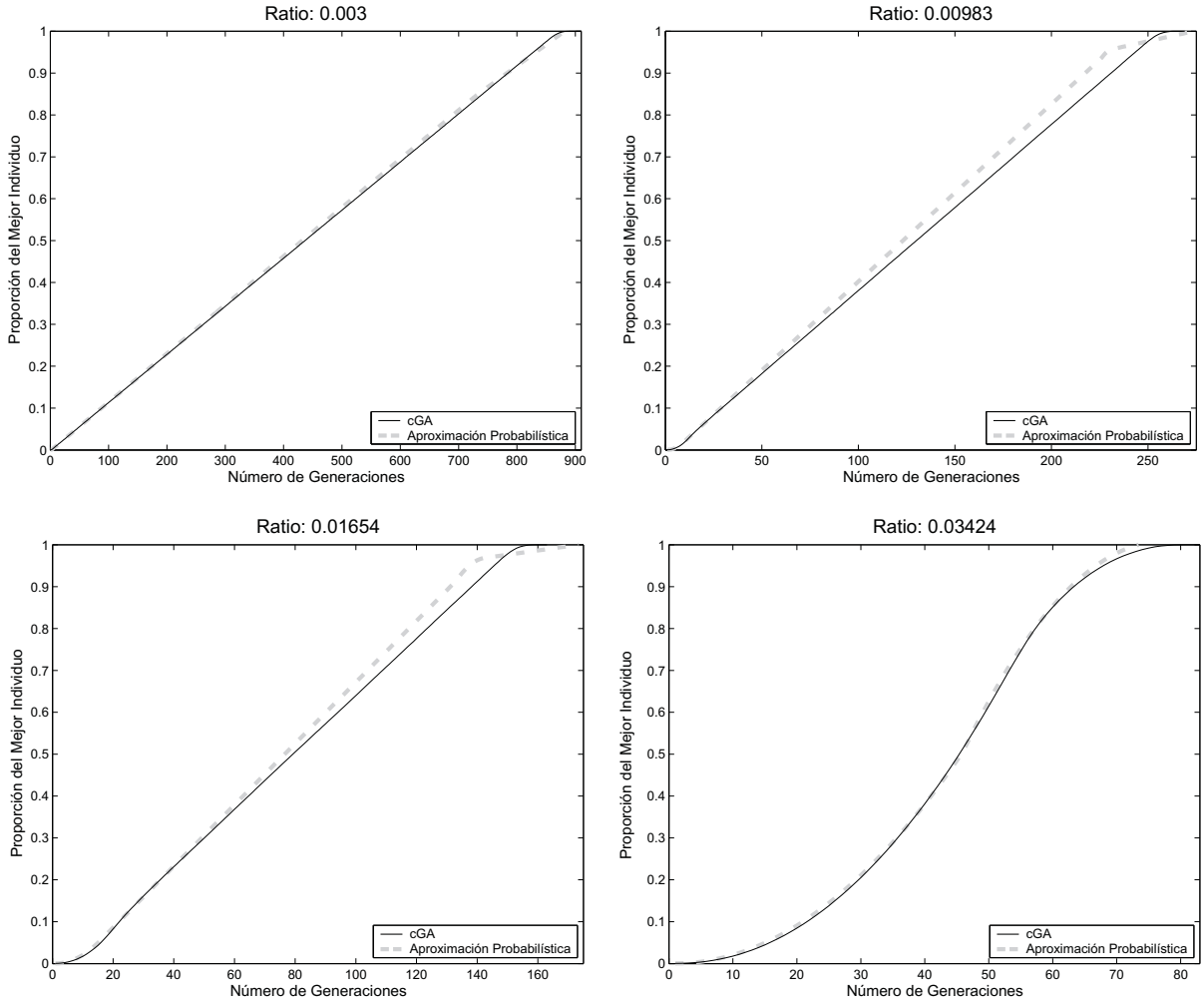


Figura 5.15: Aproximación probabilística a las curvas convergencia de cuatro cGAs. Selección por ordenación lineal.

5.2.4. Comparación de los Mejores Modelos

Tras el estudio realizado en las secciones 5.2.2 y 5.2.3, en esta sección realizamos una comparación de los dos mejores modelos analizados. Se trata del modelo de Giacobini et al. y la aproximación probabilística propuesta en este trabajo (en la Sección 5.2.3). Aunque el modelo polinómico es el que mejor se ajusta de todos los estudiados, no lo consideramos en esta sección debido a que es muy poco intuitivo, y además no contempla la utilización de otros métodos de selección distintos del torneo. Requerimos que el modelo sea lo más intuitivo posible porque debe ayudarnos a comprender el algoritmo y, aunque este modelo ajusta muy bien, no ayuda a esta tarea prioritaria. Por todo esto consideramos que ese modelo es aún susceptible de una profunda investigación.

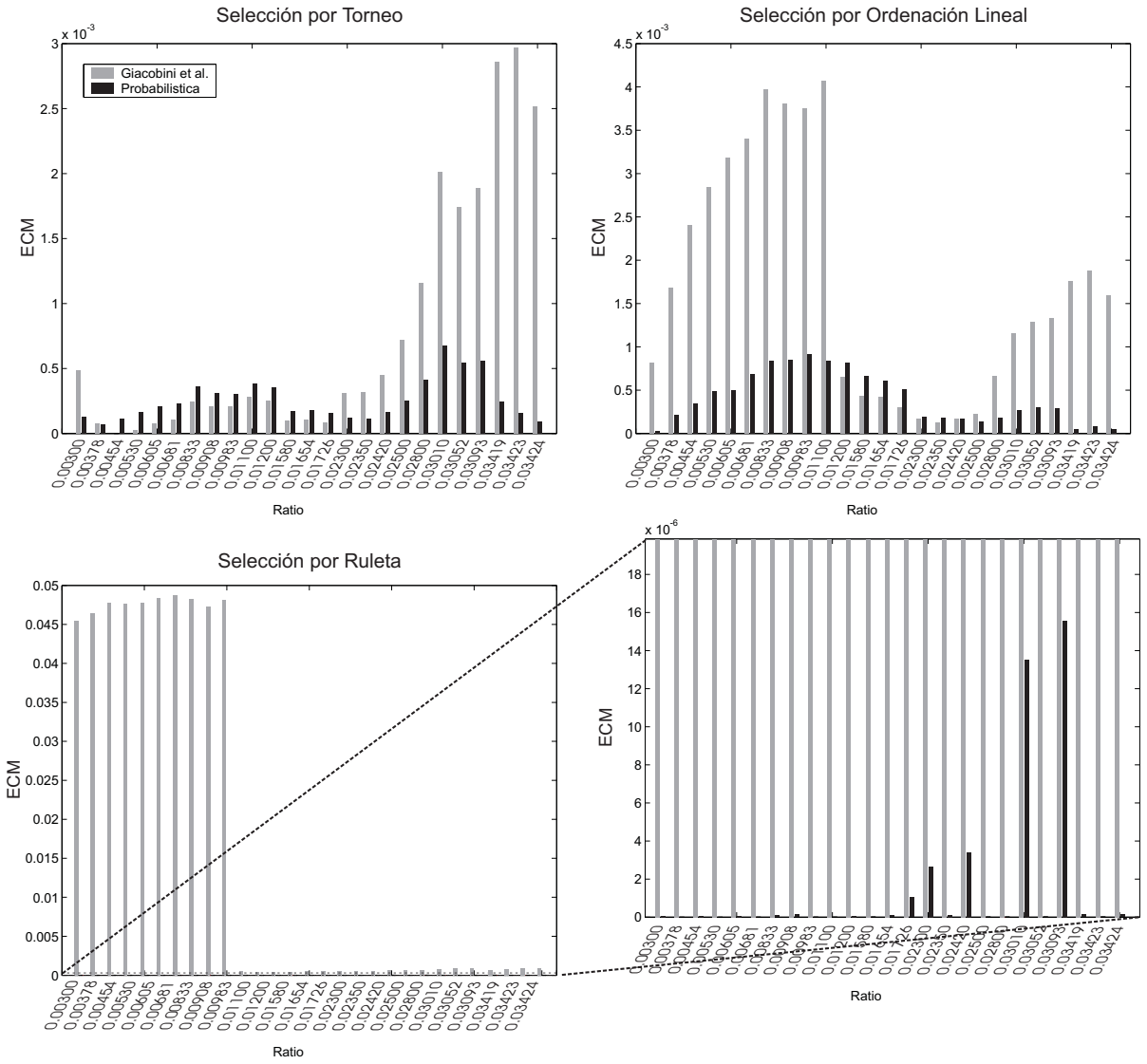


Figura 5.16: Comparación de los errores obtenidos por el modelo de Giacobini et al. y el probabilístico presentado en esta tesis.

Por tanto, se muestra en la Figura 5.16 una comparación de la recurrencia probabilística propuesta por Giacobini et al. en [119] con la aproximación probabilística propuesta en este trabajo de tesis. Se muestra el error cuadrático medio –o ECM, definido en (5.3)– cometido por ambos modelos con respecto a 75 cGAs distintos utilizando las selecciones por torneo, por ruleta y por ordenación lineal para las 25 formas de población distintas estudiadas. El valor de ratio = 0.003 se corresponde con la población más estrecha estudiada en este trabajo, formada por 4×1024 individuos, mientras que el valor 0.3424 de la ratio se corresponde con la población cuadrada (64×64 individuos).

Tabla 5.2: Comparación del modelo de Giacobini et al. y el probabilístico propuesto en este trabajo en términos del error medio cometido Δ.

Selección	**Giacobini et al.**	**Probabilístico**
Ruleta	0.0194	$\mathbf{1.4904 \cdot 10^{-6}}$
Torneo	$7.6735 \cdot 10^{-4}$	$\mathbf{2.5788 \cdot 10^{-4}}$
Ordenación Lineal	0.0017	$\mathbf{4.0705 \cdot 10^{-4}}$
Valor Medio	$7.2891 \cdot 10^{-3}$	$\mathbf{2.2214 \cdot 10^{-4}}$

Las tres gráficas mostradas en la Figura 5.16 ponen de manifiesto la clara superioridad del modelo probabilístico propuesto en este trabajo frente al de Giacobini et al. En este sentido, llama la atención la gran diferencia existente entre ambos modelos en el caso de utilizar la selección por ruleta. En cualquier caso, el modelo de Giacobini et al. es capaz de mejorar el error obtenido por el modelo probabilístico aquí presentado para algunas ratios intermedias en los casos de la selección por torneo y por ordenación lineal, aunque las diferencias son despreciables en la mayoría de los casos. En la Tabla 5.2 se muestra el error medio Δ cometido por cada modelo para las tres selecciones estudiadas, junto con la media de los tres casos. Como se puede apreciar, el modelo probabilístico es el mejor para las tres selecciones (ver cifras en **negrita**), siendo el error medio de las tres selecciones casi dos órdenes de magnitud menor que en el caso del modelo propuesto por Giacobini et al.

Si observamos el comportamiento del modelo propuesto por Giacobini et al., sorprende el mal comportamiento que muestra para algunos tipos de poblaciones en función del método de selección utilizado. En el caso del torneo binario, este modelo se comporta claramente peor para las ratios mayores (poblaciones menos estrechas), mientras que por el contrario, en el caso de la ordenación lineal y la ruleta el modelo se comporta peor para las ratios menores (poblaciones estrechas), siendo especialmente importante esta diferencia en el caso de la selección por ruleta, en el que el error es mayor que 0.045 para ratios menores que 0.00983, mientras que para el resto de las ratios estudiadas el error desciende dos órdenes de magnitud.

5.3. Estudio Práctico

Hasta este momento, hemos estudiado teóricamente el comportamiento de distintos cGAs. Para caracterizar el comportamiento teórico de los cGAs, existen algunos trabajos que han propuesto ecuaciones matemáticas que modelan la presión de selección del algoritmo. En la primera parte del capítulo se han comparado los modelos matemáticos existentes, y se han propuesto dos más nunca antes publicados.

En esta sección realizamos un estudio práctico para sustentar los resultados obtenidos en el estudio anterior. Para ello, presentamos los resultados de la comparación de cGAs síncronos y asíncronos, así como de cGAs síncronos con diferentes valores de ratio, utilizando siempre una forma de vecindario constante (L5). Nótese que el estudio completo de los problemas tratados en esta sección está fuera del ámbito de este trabajo, y tampoco es el objetivo de este estudio comparar el rendimiento de los cGAs estudiados con algoritmos del estado del arte y otros heurísticos para optimización combinatoria y numérica. Para esto, deberíamos al menos ajustar los parámetros e incluir búsqueda local en el algoritmo, que no es el caso. Por tanto, los resultados presentados en esta sección pertenecen únicamente al rendimiento relativo de los cGAs con las diferentes políticas de actualización y ratios entre ellos.

La Sección 5.3.1 está dedicada al caso discreto, mientras que en la Sección 5.3.2 nos centramos en la optimización continua.

Tabla 5.3: Parametrización utilizada en los cGAs para los problemas discretos.

Tamaño de Población	400 individuos
Vecindario	L5
Selección de los Padres	torneo binario + torneo binario
Recombinación	DPX, $p_c = 1.0$
Bit Mutación	Bit-flip, $p_m = 1/L$ ($10/L$ para FMS)
Longitud de Individuos	L
Reemplazo	Reemp_si_Mejor

5.3.1. Validación sobre Optimización Combinatoria

En esta sección presentamos y analizamos los resultados de resolver un conjunto de problemas discretos con cGAs asíncronos utilizando diversas políticas de actualización de individuos y otros cGAs síncronos utilizando distintas formas de mallas para albergar la población.

En nuestros experimentos, trabajamos con el mismo conjunto de problemas estudiado en [28], que incluye el problema masivamente multimodal (MMDP), el problema de modulación en frecuencia de sonidos (FMS), y el generador de problemas multimodales P-PEAKS. Además, hemos extendido este conjunto de problemas con el problema del diseño de códigos correctores de errores (ECC), el del máximo corte de un grafo (MAXCUT), el de las tareas de mínima espera (MTTP), y el problema de satisfacibilidad (SAT). Con el fin de hacer de esta tesis un documento autocontenido, todos estos problemas se encuentran descritos en el Apéndice A. El tamaño del problema MMDP utilizado en este estudio es $k = 20$, mientras que "mttp20" es la instancia estudiada en el caso de MTTP.

Podemos considerar que el conjunto de problemas seleccionado para este estudio es representativo ya que contiene multitud de diversas características interesantes, como son la multimodalidad, el uso de restricciones o la identificación de parámetros; además, muchas de ellas son funciones engañosas o generadoras de problemas. Estos son ingredientes importantes en cualquier trabajo que trate de evaluar aproximaciones algorítmicas con el objetivo de conseguir resultados fiables, como se afirma en el trabajo de Whitley et al. in [290].

La elección de esta batería de problemas se puede justificar, además, tanto por su alta dificultad como por sus dominios de aplicación (identificación de parámetros, telecomunicaciones, optimización combinatoria, planificación, etc.). Esto nos proporciona un elevado nivel de relevancia en nuestros resultados, aunque la evaluación de las conclusiones es consecuentemente más laboriosa que en el caso de utilizar un conjunto de problemas reducido.

A continuación, presentamos y analizamos los resultados obtenidos durante nuestras pruebas.

Análisis Experimental

Presentamos a continuación los resultados obtenidos tras resolver los problemas propuestos anteriormente con cuatro cEAs asíncronos y tres cEAs con ratios estáticas diferentes. Los resultados obtenidos pueden ser reproducidos utilizando JCell. La configuración del algoritmo para el caso de la optimización combinatoria se muestra en la Tabla 5.3. Como vemos, la población está siempre compuesta por 400 individuos. El vecindario utilizado es L5, de donde los dos padres son seleccionados utilizando el operador de torneo binario. El operador de recombinación utilizado es el del cruce de dos puntos, y la mutación consiste en cambiar el valor de cada bit con probabilidad $1/L$, siendo L la longitud de los individuos. Finalmente, el nuevo individuo generado (descendiente) reemplaza al actual sólo si es mejor, es decir, si tiene un valor de fitness mayor (asumiendo que queremos maximizar). La condición de terminación de los algoritmos es encontrar la solución óptima o realizar un número máximo de generaciones del algoritmo. Este número máximo de generaciones es distinto para cada problema (debido a sus distintas complejidades) y está especificado en el título de las tablas 5.5 a 5.11.

En la Tabla 5.4 presentamos las ratios utilizadas en este estudio. Como puede verse se trata de una población cuadrada, con ratio = 0.11 (Square), y dos poblaciones rectangulares: de 10×40 (Rectangular) y 4×100 individuos (Narrow), teniendo ratios de valor 0.075 y 0.031, respectivamente.

Mostramos en las siguientes tablas los resultados obtenidos para los problemas antes mencionados: MMDP (Tabla 5.5), FMS (Tabla 5.6), P-PEAKS (Tabla 5.7), ECC (Tabla 5.8), MAXCUT (Tabla 5.9), MTTP (Tabla 5.10), y SAT (Tabla 5.11). En estas tablas se presentan los valores medios del mejor fitness encontrado en cada ejecución, el número de evaluaciones necesario para obtener la solución óptima al problema (en el caso de que se encuentre), y el porcentaje de ejecuciones en las que el óptimo fue encontrado (tasa de éxito). Por tanto, estamos analizando la distancia final al óptimo (especialmente interesante cuando no se encuentra dicho óptimo), el esfuerzo requerido por el algoritmo, y la eficacia esperada del algoritmo, respectivamente. Con el fin de obtener resultados significativos, se han realizado 100 ejecuciones independientes de cada algoritmo para todos los problemas estudiados. Los mejores resultados para cada problema están resaltados en **negrita**.

Del análisis de estas tablas podemos obtener algunas conclusiones claras. Primero, los algoritmos asíncronos estudiados tienden a necesitar un menor número de evaluaciones que los síncronos para obtener el óptimo, en general. Además, las diferencias entre algoritmos asíncronos y síncronos son estadísticamente significativas para todos los casos (excepto en dos), lo que nos indica que las versiones asíncronas son más eficientes que los cGAs síncronos con distintas ratios. Estas dos excepciones se dan en los casos de los problemas MMDP y SAT.

Por otro lado, los algoritmos síncronos obtienen un porcentaje de soluciones encontradas (tasa de éxito) similar o incluso mejor que las versiones asíncronas, mientras que en la calidad de las soluciones encontradas no siempre existen diferencias significativas (las excepciones son, probablemente, debido a las diferencias en la tasa de éxito).

Si prestamos atención a la tasa de éxito, podemos concluir que las políticas síncronas mejoran a los algoritmos asíncronos: ligeramente en el caso de los valores de fitness encontrados, y claramente en términos de la probabilidad de encontrar la solución (es decir, frecuencia de encontrar un óptimo).

Otro resultado interesante es el hecho de que podemos definir dos tipos de problemas: aquellos para los que se encuentra la solución óptima por todos los cEAs (100 % de tasa de éxito), y aquellos en los que no se encuentra el óptimo en el 100 % de las ejecuciones por ningún algoritmo. Problemas que parecen ser tratables con cEAs directamente, o para los que es necesario algún tipo de ayuda (no estudiada aún), como por ejemplo la inclusión de búsqueda local.

Con el fin de resumir el gran conjunto de resultados generado y así obtener conclusiones útiles, presentamos una clasificación de los mejores algoritmos en función de tres métricas diferentes: media de la mejor solución encontrada, media del número de generaciones necesarias para encontrar el óptimo (cuando se encuentra), y la tasa de éxito. Estas tres clasificaciones, que se muestran en la Tabla 5.12, se han calculado sumando la posición (de mejor a peor: 1, 2, 3, ...) en la que se encuentran los algoritmos para los resultados previos presentados de la Tabla 5.5 a la Tabla 5.11, de acuerdo con los tres criterios.

Tabla 5.4: Ratios estudiadas.

Nombre	**(forma de la población)**	**Valor de la ratio**
Square	(20×20 individuos)	0.11
Rectangular	(10×40 individuos)	0.075
Narrow	(4×100 individuos)	0.031

Tabla 5.5: Problema MMDP con un máximo de 1000 generaciones.

Algoritmo	**Solución Media (Óptimo=20)**	**Media de Generaciones**	**Tasa de Éxito**
Square	19.813	214.2	57 %
Rectangular	19.824	236.1	58 %
Narrow	19.842	299.7	**61 %**
LS	19.518	343.5	23 %
FRS	19.601	209.9	31 %
NRS	19.536	**152.9**	28 %
UC	**19.615**	295.7	36 %

Tabla 5.6: Problema FMS con un máximo de 3000 generaciones.

Algoritmo	**Solución Media (Óptimo≥100)**	**Media de Generaciones**	**Tasa de Éxito**
Square	**90.46**	437.4	57 %
Rectangular	85.78	404.3	61 %
Narrow	80.76	610.9	**63 %**
LS	81.44	**353.4**	58 %
FRS	73.11	386.2	55 %
NRS	76.21	401.5	56 %
UC	83.56	405.2	57 %

Tabla 5.7: Problema P-PEAKS con un máximo de 100 generaciones.

Algoritmo	**Solución Media (Óptimo=1)**	**Media de Generaciones**	**Tasa de Éxito**
Square	**1.0**	51.8	**100 %**
Rectangular	**1.0**	50.4	**100 %**
Narrow	**1.0**	**53.9**	**100 %**
LS	**1.0**	34.8	**100 %**
FRS	**1.0**	38.4	**100 %**
NRS	**1.0**	38.8	**100 %**
UC	**1.0**	40.1	**100 %**

Tabla 5.8: Problema ECC con un máximo de 500 generaciones.

Algoritmo	**Solución Media (Óptimo=730)**	**Media de Generaciones**	**Tasa de Éxito**
Square	0.0670	93.9	85 %
Rectangular	0.0671	93.4	88 %
Narrow	**0.0673**	104.2	**94 %**
LS	0.0672	79.7	89 %
FRS	0.0672	82.4	90 %
NRS	0.0672	**79.5**	89 %
UC	0.0671	87.3	86 %

Tabla 5.9: Problema MAXCUT con un máximo de 100 generaciones.

Algoritmo	**Solución Media (Óptimo=56.74)**	**Media de Generaciones**	**Tasa de Éxito**
Square	**56.74**	11.3	**100 %**
Rectangular	**56.74**	11.0	**100 %**
Narrow	**56.74**	11.9	**100 %**
LS	**56.74**	**9.5**	**100 %**
FRS	**56.74**	9.7	**100 %**
NRS	**56.74**	9.6	**100 %**
UC	**56.74**	9.6	**100 %**

Tabla 5.10: Problema MTTP con un máximo de 50 generaciones.

Algoritmo	**Solución Media (Ópt.=0.02439)**	**Media de Generaciones**	**Tasa de Éxito**
Square	**0.02439**	8.4	**100 %**
Rectangular	**0.02439**	8.3	**100 %**
Narrow	**0.02439**	8.9	**100 %**
LS	**0.02439**	**5.9**	**100 %**
FRS	**0.02439**	6.2	**100 %**
NRS	**0.02439**	6.3	**100 %**
UC	**0.02439**	6.3	**100 %**

Tabla 5.11: Problema SAT con un máximo de 3000 generaciones.

Algoritmo	**Solución Media (Óptimo=430)**	**Media de Generaciones**	**Tasa de Éxito**
Square	429.54	703.1	79 %
Rectangular	**429.67**	706.3	84 %
Narrow	429.61	763.7	81 %
LS	429.52	**463.2**	78 %
FRS	**429.67**	497.7	**85 %**
NRS	429.49	610.5	75 %
UC	429.50	725.5	76 %

Como podríamos esperar tras los resultados anteriores, de acuerdo con los criterios de la media de la mejor solución encontrada y la tasa de éxito, los algoritmos síncronos con cualquiera de las tres ratios estudiadas son en general más precisos que todos los asíncronos para los problemas utilizados en nuestras pruebas, con una posición especialmente privilegiada para la población Narrow. Por otro lado, las versiones asíncronas mejoran claramente a los algoritmos síncronos en términos de la media de las generaciones necesarias para encontrar la solución (eficiencia), con cierta tendencia hacia LS como la mejor política asíncrona en el caso de los problemas discretos.

Tabla 5.12: Clasificación de los mejores algoritmos para los problemas discretos.

Solución Media		**Media de Generaciones**		**Tasa de Éxito**	
1 Narrow	10	1 LS	14	1 Narrow	6
1 Rectangular	10	2 NRS	16	2 Rectangular	10
3 Square	14	3 FRS	18	3 FRS	14
4 FRS	15	4 UC	30	4 LS	15
5 LS	18	5 Rectangular	33	5 Square	17
5 UC	18	6 Square	37	6 UC	19
7 NRS	21	7 Narrow	48	7 NRS	21

5.3.2. Validación sobre Optimización Continua

Extenderemos en esta sección el trabajo de la anterior mediante la aplicación de los mismos modelos algorítmicos a algunas funciones continuas con el fin de obtener un estudio más extensivo. Este estudio puede ser interesante para analizar el comportamiento de los algoritmos en optimización continua, con la posibilidad de contrastarlo con el caso discreto. Las funciones que hemos seleccionado para este estudio son tres funciones multimodales típicas en conjuntos de problemas numéricos; son las funciones de Rastrigin (con dimensión 10), Ackley y Fractal (para una descripción de los problemas consulte el Apéndice A). Para resolver estos tres problemas, debemos utilizar individuos con codificación real, mientras que en la sección anterior se utilizaron individuos de codificación binaria. Esta es la causa de nuestro especial interés en la experimentación con una optimización global más tradicional. La codificación que hemos empleado para estos tres problemas se ha hecho siguiendo la implementación propuesta por Michalewicz [208].

Análisis Experimental

En esta sección estudiaremos los resultados de nuestros experimentos con los problemas continuos propuestos, de la misma forma que hiciéramos anteriormente para el caso discreto. Mantenemos en este caso las ratios utilizadas en los algoritmos síncronos con respecto a las estudiadas en la Sección 5.3.1, al igual que en el caso de las políticas asíncronas. Por otro lado, se ha necesitado una configuración especial para los operadores genéticos y sus probabilidades de aplicación en el caso de los problemas de codificación real. Esta parametrización se detalla en la Tabla 5.13. Como se ve, el tamaño de la población, el del vecindario, la política de selección de los padres, y el reemplazo utilizado en este caso son los mismos que en el caso discreto. Cambian en el dominio continuo los operadores de recombinación y mutación utilizados, que son el cruce aritmético (AX) y la mutación uniforme, respectivamente; operadores típicamente utilizados en optimización continua.

Tabla 5.13: Parametrización utilizada en los cGAs para los problemas continuos.

Tamaño de la Población	400 individuos
Vecindario	L5
Selección de los Padres	torneo binario + torneo binario
Recombinación	AX, $p_c = 1.0$
Mutación de Genes	Uniforme, $p_m = 1/2L$
Longitud de Individuos	L
Reemplazo	Reemp_si_Mejor

Presentamos en las tablas 5.14 a 5.16 los resultados de nuestros experimentos con los problemas Rastrigin (Tabla 5.14), Ackley (Tabla 5.15), y Fractal (Tabla 5.16). Al igual que en el caso de los problemas discretos, estas tablas contienen valores de la media de las mejores soluciones encontradas, el número medio de generaciones necesarias para encontrar el óptimo, y la tasa de éxito. Estos tres valores han sido calculados sobre 100 ejecuciones independientes. Para estos tres problemas de codificación real, se considera que la búsqueda termina con éxito cuando un individuo alcanza el óptimo con un error del 10 %.

Tabla 5.14: Problema Rastrigin con un máximo de 700 generaciones.

Algoritmo	**Solución Media (mejor≤ 0.1)**	**Media de Generaciones**	**Tasa de Éxito**
Square	0.0900	323.8	**100 %**
Rectangular	0.0883	309.8	**100 %**
Narrow	**0.0855**	354.2	**100 %**
LS	0.0899	**280.9**	**100 %**
FRS	0.0900	289.6	**100 %**
NRS	0.0906	292.2	**100 %**
UC	0.0892	292.4	**100 %**

Tabla 5.15: Problema Ackley con un máximo de 500 generaciones.

Algoritmo	**Solución Media (mejor≤ 0.1)**	**Media de Generaciones**	**Tasa de Éxito**
Square	0.0999	321.7	78 %
Rectangular	0.0994	293.1	73 %
Narrow	0.1037	**271.9**	65 %
LS	**0.0932**	302.0	84 %
FRS	0.0935	350.6	**92 %**
NRS	0.0956	335.5	87 %
UC	0.0968	335.0	85 %

Tabla 5.16: Problema Fractal con un máximo de 100 generaciones.

Algoritmo	**Solución Media (mejor≤ 0.1)**	**Media de Generaciones**	**Tasa de Éxito**
Square	0.0224	75.2	94 %
Rectangular	0.0359	62.8	78 %
Narrow	0.1648	**14.6**	16 %
LS	0.0168	69.7	98 %
FRS	0.0151	71.5	**100** %
NRS	0.0163	73.6	98 %
UC	**0.0138**	72.8	96 %

Los resultados que hemos obtenido con los problemas continuos no son tan claros como en el caso discreto. Con respecto al número medio de generaciones necesarias para encontrar una solución óptima, los algoritmos asíncronos no realizan siempre un menor número de generaciones que los síncronos; dos ejemplos son las funciones de Ackley y Fractal. Las diferencias entre los algoritmos síncronos y asíncronos son generalmente significativas. Esto nos indica una mayor eficiencia de los cGAs con distintas ratios, resultado que contrasta con las conclusiones desprendidas del caso de los problemas discretos. Por otro lado, en contra de lo observado en el caso discreto, las tasas de éxito obtenidas por los algoritmos asíncronos son mayores que en el caso de los cGAs síncronos en general. Además, en contra de los resultados de la Sección 5.3.1, donde o bien todos los algoritmos encuentran el óptimo en el 100 % de las ejecuciones o bien ninguno es capaz de encontrarlo, el cGA asíncrono con la política de actualización FRS es el único algoritmo capaz de encontrar la solución en todas las ejecuciones del problema Fractal.

Tabla 5.17: Clasificación de los algoritmos en los problemas continuos.

Solución Media		**Media de Generaciones**		**Tasa de Éxito**	
1 UC	8	1 LS	7	1 FRS	3
2 LS	9	2 Narrow	9	5 NRS	5
2 FRS	9	2 Rectangular	9	4 LS	7
4 NRS	13	4 FRS	13	6 UC	8
4 Rectangular	13	5 UC	14	7 Square	11
6 Narrow	15	6 NRS	15	3 Rectangular	13
7 Square	16	7 Square	17	2 Narrow	15

Con el fin de resumir estos resultados, y siguiendo la estructura de la Sección 5.3.1, presentamos en la Tabla 5.17 una clasificación con los mejores algoritmos en todos los problemas en términos de la media de la mejor solución conocida, el número medio de generaciones necesario para encontrar el óptimo, y la tasa de éxito. Se puede ver en esta tabla cómo existe una tendencia de los algoritmos asíncronos a ser mejores que los síncronos en términos de la media de la mejor solución encontrada y la tasa de éxito, mientras que los algoritmos síncronos con diferentes ratios parecen ser más eficientes que la mayoría de los asíncronos en general (las excepciones son el cGA síncrono con malla cuadrada –Square– y el asíncrono LS).

5.4. Conclusiones

En la primera parte de este capítulo hemos estudiado la presión de selección inducida por varias políticas asíncronas de actualización de individuos en cGAs, así como por distintos valores de ratio (distintas formas de la población) utilizados en cGAs síncronos. Todos estos algoritmos estudiados realizan una búsqueda diferente en el espacio de soluciones. Como hemos visto, se puede ajustar la presión de selección de un cGA mediante la elección de una política de actualización y/o un valor de ratio entre la malla de la población y el vecindario sin tener que tratar con el ajuste de parámetros numéricos adicionales del algoritmo. Esto es un aspecto importante de los algoritmos que permite al usuario utilizar conocimiento ya existente en lugar de inventar una nueva clase de heurística.

Nuestra conclusión es que los cGAs pueden ser inducidos fácilmente a promocionar la exploración o explotación simplemente cambiando la política de actualización de la ratio entre la población y el vecindario. Esto abre nuevas líneas de investigación para decidir formas eficientes de cambiar de una política o una ratio dadas a otra para llegar a la solución óptima con un menor esfuerzo que en el caso del cGA básico u otros tipos de GAs.

En una segunda parte del capítulo tratamos de dar algunos modelos matemáticos que nos permitan cuantificar de antemano mediante ecuaciones matemáticas la presión de selección de los cGAs. Se han estudiado todos los modelos existentes en la literatura, y se proponen dos más en este capítulo (probabilístico y polinómico). Los dos mejores métodos de entre los estudiados coinciden en ser los únicos dos modelos basados en probabilidades. Comparando estos dos modelos, obtenemos que nuestra propuesta mejora claramente a la previamente existente en la literatura.

Por último, en una tercera parte del capítulo hemos aplicado nuestros cGAs extendidos a un conjunto de problemas que pertenecen a los dominios de la optimización combinatoria y continua. Aunque nuestro objetivo no era obtener técnicas capaces de competir con heurísticas especializadas del estado del arte, los

resultados muestran claramente que los cGAs son técnicas de optimización muy eficientes, que pueden ser mejoradas aún más mediante la hibridación con técnicas de búsqueda local [17, 18, 19, 104]. Los resultados en los problemas de test confirman claramente, con algunas pequeñas excepciones, que la capacidad de estos algoritmos para resolver los problemas utilizando distintas ratios y políticas de actualización están directamente relacionadas con la presión de selección de los algoritmos, mostrando que la explotación juega un papel muy importante en la búsqueda. Está claro que el papel de la exploración debería ser más importante en problemas aún más difíciles que los estudiados, pero este aspecto se puede tratar en nuestros algoritmos utilizando ajustes de parámetros más explorativos, además de utilizando diferentes estrategias de cGAs en distintos tiempos durante la búsqueda de forma dinámica [13, 15].

Nuestros resultados son claros: con respecto a los problemas discretos, los algoritmos asíncronos son más eficientes que los síncronos; con diferencias estadísticamente significativas para la mayoría de los problemas. Por otro lado, si prestamos atención a los otros dos aspectos estudiados, podemos concluir que las políticas síncronas con diferentes ratios mejoran a los algoritmos asíncronos: ligeramente en términos de la media de las soluciones encontradas, y claramente en términos de la probabilidad de encontrar una solución (es decir, la frecuencia de encontrar el óptimo).

Por el contrario, si prestamos atención a los experimentos realizados sobre problemas continuos podemos obtener conclusiones diferentes (de alguna forma complementarias). Los algoritmos asíncronos mejoran a los síncronos en los casos de la media de las soluciones encontradas y la tasa de éxito, mientras que en el caso del número medio de generaciones realizadas los algoritmos síncronos son en general más eficientes que los asíncronos.

Parte II

Metodología

Capítulo 6

Diseño de cGAs Adaptativos

En el Capítulo 5 se ha estudiado teóricamente que el uso de distintas ratios y políticas de actualización en cGAs inducen una diferente presión de selección en el algoritmo. Al extender este estudio de forma empírica tratando con la resolución de problemas complejos, se comprobó que la presión de selección de los algoritmos tiene efecto en el comportamiento de la búsqueda que realiza el cGA al resolver un problema. En las conclusiones de dicho capítulo se muestra que la explotación juega un papel muy importante en la búsqueda del algoritmo, y se insinúa que podemos potenciar la faceta explorativa de un cGA utilizando diferentes estrategias en distintos tiempos durante la búsqueda de forma dinámica.

En este capítulo presentamos un nuevo modelo de cGA que adapta automáticamente durante la ejecución las capacidades de exploración y explotación de la búsqueda que realiza. Para ello, modifica la ratio entre los radios de vecindario y población en función de los cambios sufridos en la diversidad de la población durante las distintas generaciones.

La estructura de este capítulo se describe a continuación. En la siguiente sección se presenta una breve justificación al uso de poblaciones adaptativas en cGAs. En la Sección 6.2 se resume brevemente el estado del arte en algoritmos auto-adaptativos. La Sección 6.3 presenta diferentes cGAs, caracterizados por sus poblaciones de ratios estáticas, pre-programadas, y adaptativas, y que serán analizados y comparados en la Sección 6.4. Este estudio se extiende en la Sección 6.5 con el análisis del comportamiento de los cGAs adaptativos con políticas asíncronas de actualización de individuos. Finalmente, el capítulo termina con un resumen de nuestras principales conclusiones (Sección 6.6).

6.1. Introducción

Según estudiamos en el Capítulo 5, el uso de poblaciones más estrechas (ratios menores) favorece la exploración del algoritmo, mientras que una población menos estrecha potencia la explotación en el espacio de búsqueda. Como trabajamos con un vecindario constante (usamos L5), el valor de la ratio dependerá únicamente del radio de la malla. Este radio se incrementará cuanto más estrecha sea la malla, reduciéndose el valor de la ratio resultante.

Reducir la ratio es equivalente a reducir la intensidad de selección global, promoviendo la exploración. Esto nos permitirá mantener una diversidad más elevada en el algoritmo, lo que mejora los resultados en problemas difíciles (por ejemplo problemas multimodales y epistáticos). Por otro lado, la búsqueda que se realiza dentro de cada vecindario guía al algoritmo hacia la explotación de la población. En

este capítulo investigamos cómo influye esta ratio en la eficiencia de la búsqueda sobre una variedad de dominios. Cambiar la ratio durante la búsqueda es una característica única de los cGAs que puede ser utilizada ventajosamente para cambiar de exploración a explotación con una complejidad mínima y sin introducir un nuevo tipo de algoritmo ni hibridaciones en el funcionamiento canónico.

Existen multitud de técnicas para manejar la relación entre exploración y explotación. Entre ellas, merece la pena hacer una mención especial a los algoritmos heterogéneos [6, 23, 142], en los que se ejecutan varios algoritmos con características similares en múltiples sub-poblaciones que colaboran con el fin de evitar la convergencia prematura. Una alternativa diferente es la utilización de algoritmos meméticos [211, 225], en los que se combina la búsqueda local con los operadores genéticos con el fin de promover la explotación local.

En el caso de los cGAs, es realmente fácil incrementar la diversidad en la población simplemente cambiando la forma de la rejilla y realojando a los individuos, como se muestra en la Sección 6.3. La razón es que el modelo celular genera restricciones en la distancia para el emparejamiento de soluciones debido al uso de vecindarios (una solución sólo podrá emparejarse con uno de sus vecinos). Por tanto, un cGA puede verse como un algoritmo con restricción en los emparejamientos basado en la distancia Euclídea.

6.2. Algunas Nociones de Auto-Adaptación

Una contribución importante de este capítulo es la definición de cGAs capaces de auto-adaptar su balance entre exploración y explotación. En esta sección haremos una introducción al campo de la adaptación y auto-adaptación en algoritmos evolutivos.

Primero recordaremos una clasificación de EAs adaptativos bien conocida [38, 145] (consulte la Figura 6.1 para más detalle). Podemos hacer una clasificación del tipo de adaptación según el mecanismo utilizado para este propósito; en concreto, se le presta atención al hecho de si se utiliza o no retroalimentación en el EA:

- *Ajuste de Parámetros.* Se da cuando los parámetros estratégicos tienen valores constantes a lo largo de toda la ejecución del EA (no existe adaptación). Consecuentemente, es necesario un agente o mecanismo externo (por ejemplo, una persona o un programa) para ajustar los parámetros deseados y elegir los valores más apropiados. Estudiamos en este capítulo algunos algoritmos con parámetros ajustados a mano.

- *Dinámica.* La adaptación dinámica se da en el caso de que exista algún mecanismo que modifica un parámetro de la estrategia de búsqueda sin ningún tipo de control externo. La clase de EA que utiliza adaptación dinámica puede ser subdividida en tres clases distintas, que se diferencian según el mecanismo de la adaptación:

  - *Determinista.* La adaptación dinámica determinista se da si el valor del parámetro de la estrategia de búsqueda se cambia según alguna regla determinista. Esta regla modifica el parámetro de la estrategia de forma determinista sin utilizar retroalimentación del EA. Ejemplos de este tipo de adaptación son los criterios pre-programados que propusieron Alba et al. [28], y que empleamos en este capítulo.
  - *Adaptativo.* La adaptación dinámica consiste en utilizar algún tipo de retroalimentación del EA con el fin de establecer la dirección y/o la magnitud del cambio en los parámetros de la estrategia de búsqueda (este tipo de adaptación se utiliza también en este capítulo).

- *Auto-adaptación*. La idea de la "evolución de la evolución" puede utilizarse para implementar la auto-adaptación de parámetros. En este caso, los parámetros que serán adaptados están codificados en la representación de las soluciones (en el cromosoma) y sufre los efectos de la mutación y la recombinación [30].

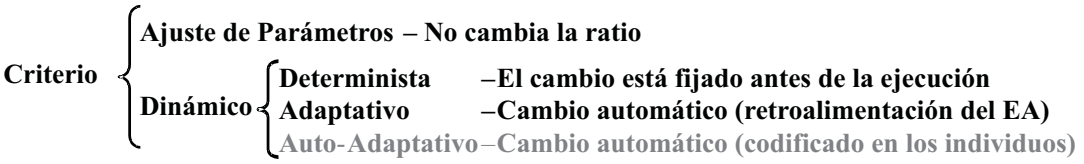


Figura 6.1: Taxonomía de los modelos de adaptación para EAs en términos de la ratio. El caso auto-adaptativo (en color gris) es el único no considerado en este capítulo.

Podemos distinguir entre diferentes familias de EAs dinámicos atendiendo al nivel en el que operan los parámetros adaptativos: el entorno (adaptación del individuo como una respuesta a cambios en el entorno –por ejemplo, términos de penalización en la función de adecuación–), población (parámetros globales a la población completa), individuos (parámetros contenidos dentro de un individuo), y componentes (parámetros de la estrategia locales a algún componente o gen del individuo). Estos niveles de adaptación pueden utilizarse con cada uno de los tipos de adaptación dinámica; además, se puede utilizar en un mismo EA una mezcla de niveles y tipos de adaptación, dando lugar a algoritmos de difícil clasificación.

Esta taxonomía de adaptación guiará al lector en la clasificación del tipo de algoritmos con los que tratamos en este capítulo. Esto proporciona un contexto apropiado para discutir las ideas incluidas en las secciones siguientes.

6.3. Descripción de los Algoritmos Utilizados

El rendimiento de un cGA puede cambiar en función de varios parámetros. Entre ellos, prestaremos especial atención a la ratio, definida como la relación entre los radios de vecindario y población, como se define en el Capítulo 2. Nuestro objetivo es estudiar los efectos de esta ratio en el comportamiento del algoritmo. Como utilizamos siempre el mismo vecindario (L5), este estudio de la ratio se reduce al análisis de los efectos de utilizar diferentes formas de la población.

Comenzamos considerando tres formas de población estáticas. Después, estudiamos dos métodos para cambiar el valor de la ratio durante la ejecución con el fin de promover la exploración o la explotación en ciertos pasos de la búsqueda. La primera aproximación a esta idea consiste en modificar el valor de la ratio en un punto fijo (predefinido) de la ejecución. Con este fin, estudiamos dos criterios distintos (originalmente propuestos en [28]): (i) cambiar de exploración a explotación, y (ii) cambiar de explotación a exploración. En el caso de la segunda aproximación, proponemos una auto-manipulación dinámica de la ratio en función del proceso de evolución.

En la Figura 6.2 se muestra una idealización teórica de la evolución de la ratio en las tres clases diferentes de algoritmos que estudiamos en este capítulo. Podemos ver cómo los algoritmos estáticos (Figura 6.2a) mantienen constante el valor de la ratio a lo largo de toda la evolución, mientras que los otros dos tipos de algoritmos lo cambian. Para los algoritmos pre-programados (Fig. 6.2b), este cambio en la ratio se realiza en un punto de la ejecución fijado *a priori*, en contraste con los algoritmos adaptativos (Fig. 6.2c), en los que la ratio cambia automáticamente durante la búsqueda en función de la velocidad de convergencia de la población.

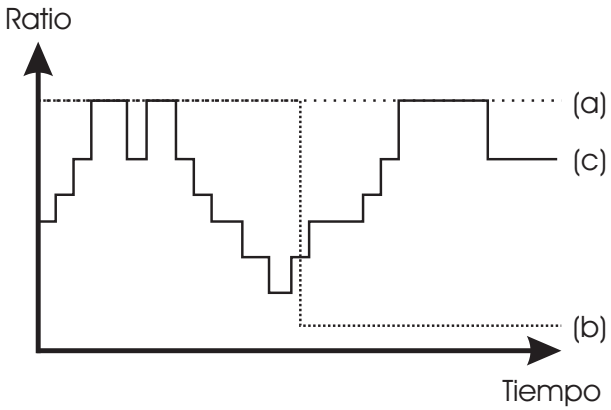


Figura 6.2: Evolución idealizada de la ratio para los criterios estático (a), pre-programado (b), y adaptativo (c).

Todos los algoritmos estudiados en este capítulo (mostrados en la Figura 6.3) se obtienen del mismo cGA canónico utilizando únicamente diferentes maneras de cambiar la ratio entre los radios del vecindario y la topología de la población. Estudiamos la influencia de esta ratio sobre una familia representativa de problemas no triviales. Esta familia de problemas está extendida del conjunto inicial incluido en el Capítulo 5, donde se establece una relación entre baja/alta ratio y exploración/explotación del algoritmo. Con el fin de ayudar al lector a comprender los resultados, explicamos en la Sección 6.3.1 los algoritmos empleados en ese trabajo. Tras esto, introducimos una nueva propuesta algorítmica adaptativa con el objetivo de evitar que el investigador tenga que realizar una definición *ad hoc* de la ratio, que mejorará (esperanzadoramente) la eficiencia del algoritmo (Sección 6.3.2).

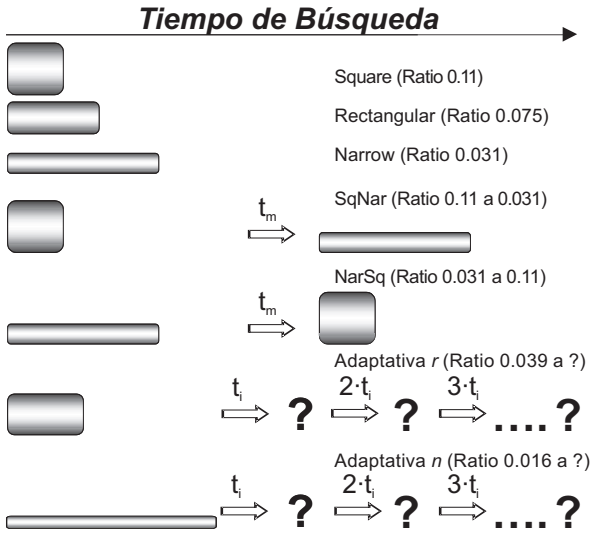


Figura 6.3: Algoritmos estudiados en este trabajo.

6.3.1. Algoritmos Estáticos y Pre-Programados

En esta sección presentamos cinco algoritmos diferentes que fueron propuestos inicialmente en [28] y que utilizaremos en nuestro estudio. Tres de ellos utilizan ratios estáticas, mientras que los otros dos implementan ratios pre-programadas (para más información consulte la Figura 6.3). Nuestro primer objetivo consiste en extender este estudio preliminar con un conjunto mayor de problemas más complejos.

En primer lugar, tomamos tres cGAs en los que se utilizan ratios estáticas bien diferentes (recuérdese que siempre utilizamos el vecindario NEWS):

- *Square*: Ratio = 0.110 (20×20 individuos).
- *Rectangular*: Ratio = 0.075 (10×40 individuos).
- *Narrow*: Ratio = 0.031 (4×100 individuos).

El tamaño total de la población es en todo caso de 400 individuos, estructurados en tres formas de malla distintas, cada una teniendo diferentes capacidades de exploración/explotación. Además, hemos utilizado dos criterios más con un cambio de ratio dinámico, aunque pre-programado *a priori*. Estos dos criterios cambian la ratio del algoritmo de forma distinta en un punto fijo de la ejecución. En nuestro caso, este cambio se realiza en el "medio" de una ejecución "típica" (t_m). Definimos el "medio" de una ejecución típica (t_m) como el punto en el que el algoritmo alcanza la mitad del número medio de evaluaciones necesario para resolver el problema con la rejilla cuadrada tradicional. Los algoritmos pre-programados estudiados cambian la ratio de 0.110 a 0.031 (al que llamamos *SqNar* –*Square* a *Narrow*–) y de 0.031 a 0.110 (llamado *NarSq* –*Narrow* a *Square*–).

Como el cambio entre exploración y explotación se realiza mediante el cambio de la forma de la población (y por tanto su ratio) en este trabajo, nosotros consideramos teóricamente que la población es una lista de individuos de longitud $m \cdot n$, de forma que la primera línea del malla de $m \cdot n$ individuos está compuesta por los n primeros individuos de la lista, la segunda fila de la rejilla se compone de los siguientes n individuos, y así sucesivamente. Por tanto, cuando realizamos el cambio de una rejilla $m \cdot n$ a otra nueva $m' \cdot n'$ (cumpliéndose que $m \cdot n = m' \cdot n'$), el individuo que se encuentre localizado en la posición (i, j) será realojada como sigue:

$$(i, j) \rightarrow ([i * n + j] \; div \; n', \; [i * n + j] \; mod \; n') \; . \tag{6.1}$$

Llamamos a este método de redistribución *contiguo*, ya que la nueva rejilla de la población se rellena siguiendo el orden de aparición de los individuos en la lista. Se muestra en la Figura 6.4 cómo se realojan en la población el individuo considerado y su vecindario cuando la forma de la población cambia de 8×8 a 4×16, siguiendo la fórmula descrita en la Ecuación 6.1. El lector puede ver como en general una parte del vecindario de cada individuo se mantiene, mientras otra parte puede cambiar. En la Figura 6.4 se ha dibujado la relocalización del individuo en la posición $(2, 4)$ y sus vecinos. Cuando la forma de la población cambia, ese individuo es realojado en la posición $(1, 4)$, cambiando sus vecinos de las posiciones del norte y del sur, y manteniendo próximos los localizados al este y el oeste. En realidad, se puede ver este cambio en la forma de la rejilla como un tipo de migración de individuos entre vecindarios, lo que introducirá una diversidad adicional en la población para las generaciones venideras.

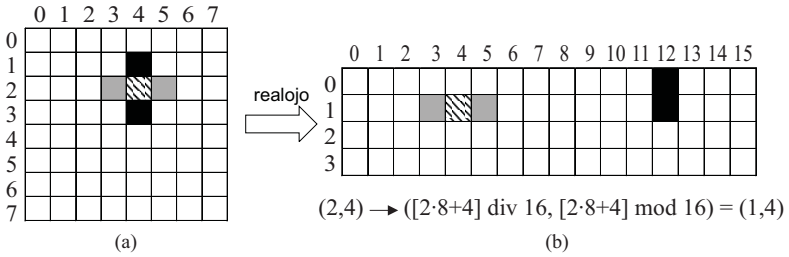


Figura 6.4: Relocalización de un individuo y sus vecinos cuando la rejilla de la población cambia de (a) 8×8 a otra (b) con forma 4×16.

6.3.2. Algoritmos Adaptativos

Presentamos en esta sección una nueva aportación al campo de la auto-adaptación. La idea principal, como en la mayoría de los algoritmos adaptativos, es el uso de algún mecanismo de retroalimentación que monitorice la evolución y que de forma dinámica premie o penalice el valor de ciertos parámetros en función de su impacto en la calidad de las soluciones. Un ejemplo de estos mecanismos es el método de Davis [72] que, con el fin de producir una mejora en el fitness de los individuos, adapta las probabilidades de aplicación de los operadores en GAs basados en el éxito o fracaso observados. Otros ejemplos son las aproximaciones de [31] y [249] para adaptar el tamaño de las poblaciones bien asignando tiempos de vida a los individuos basados en su fitness o estableciendo una competición entre las sub-poblaciones basada en los fitness de los mejores miembros de la población.

Una diferencia principal en nuestro caso es que la monitorización de la realimentación y las acciones tomadas son muy poco costosas computacionalmente. Creemos realmente que ésta debe ser la característica principal que cualquier algoritmo adaptativo debe tener con el fin de ser útil para otros investigadores. La literatura contiene numerosos ejemplos de algoritmos adaptativos interesantes cuyo control de la retroalimentación o las acciones adaptativas son tan costosas computacionalmente que muchos otros métodos u operaciones ajustadas a mano podrían resultar en algoritmos más eficientes o fáciles de utilizar.

Los mecanismos adaptativos que proponemos en este trabajo se muestran en la Figura 6.3 (analizamos varios mecanismos, no sólo uno). Estos mecanismos modifican la forma de la rejilla durante la búsqueda en función de la velocidad de convergencia. Una ventaja importante de estos criterios adaptativos consiste en que no es necesario establecer la forma de la población a ningún valor *ad hoc*, ya que se realiza un cálculo dinámico de cuál es la forma más apropiada con una cierta frecuencia (t_i) durante la búsqueda. Esto además ayuda a reducir la sobrecarga al mínimo. El Algoritmo 6.1 describe el patrón básico adaptativo utilizado; C_1 y C_2 representan las medidas de la velocidad de convergencia a utilizar.

Algoritmo 6.1 Patrón para los criterios dinámicos adaptativos propuestos.

1. **si** C_1 **entonces**
2. *CambiaA*(más_cuadrado) // explotar
3. **en otro caso si** C_2 **entonces**
4. *CambiaA*(más_estrecho) // explorar
5. **en otro caso**
6. *No_Cambiar*
7. **fin si**

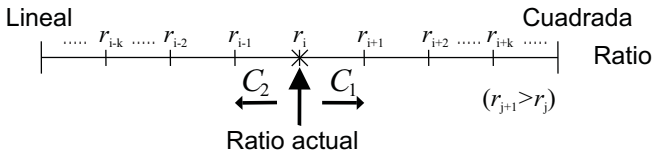


Figura 6.5: Cambio en la ratio realizado cuando se cumplen las condiciones C_1 y C_2 del Algoritmo 6.1.

Los criterios adaptativos tratan de incrementar la explotación local cuando el algoritmo evoluciona muy lentamente, es decir, cuando la velocidad de convergencia cae bajo un valor umbral dado ($\varepsilon \in [0, 1]$). Esto se consigue cambiando la forma de la rejilla a la siguiente más cuadrada, como se representa mediante la condición de la línea 1 en el Algoritmo 6.1. Si, por otro lado, la búsqueda se desarrolla con demasiada velocidad, podríamos perder diversidad rápidamente, existiendo por tanto un riesgo elevado de que la búsqueda quede atascada en un óptimo local. En este segundo caso la forma de la población se cambiará a una rejilla más estrecha para promocionar la exploración y la diversidad en las siguientes generaciones. La condición para detectar esta situación se expresa en la línea 3 del Algoritmo 6.1. Si no se cumple ninguna de las dos condiciones (ni C_1 ni C_2), la forma de la población permanece constante (línea 6).

Como se puede ver, éste es un patrón de búsqueda genérico, y se pueden utilizar múltiples criterios de control (C_i) para decidir si el algoritmo debería potenciar la exploración o la explotación de la población para las siguientes t_i generaciones.

El objetivo de estos criterios adaptativos es el de mantener la diversidad en la población y encaminar la búsqueda durante la ejecución hacia el óptimo global. Nótese que un cambio en la forma de la rejilla implica la relocalización de los individuos, que es un tipo de migración, ya que los individuos que estaban en distintas zonas se pueden volver vecinos tras el cambio. Por tanto, este cambio en la topología es también una fuente adicional para incrementar la diversidad, intrínseca del criterio adaptativo.

Cuando se cumple el criterio, el patrón de búsqueda adaptativa realiza un cambio en la forma de la rejilla. La función **CambiaA**(más_cuadrado/más_estrecho) del Algoritmo 6.1 cambia la rejilla a la siguiente forma de rejilla permitida inmediatamente más cuadrada/estrecha. Los casos extremos son la rejilla cuadrada y la completamente lineal (en forma de anillo), como se muestra en la Figura 6.5. Cuando la rejilla actual es la cuadrada y el algoritmo necesita cambiar a la siguiente rejilla "más cuadrada", o cuando la rejilla actual es la lineal y es necesario el cambio a una rejilla "más estrecha", el algoritmo no realiza ningún cambio en la población, manteniendo la forma constante. En el resto de los casos, cuando el cambio es posible, se realiza calculando la nueva posición para cada individuo localizado en la posición (i, j) como se muestra en la Ecuación 6.1. Por supuesto, se podrían haber utilizado otros cambios, pero, en nuestra búsqueda de la eficiencia, consideramos que el método propuesto induce una sobrecarga despreciable.

Una vez hemos fijado el patrón de búsqueda adaptativo y las reglas para modificar la forma de la rejilla, procedemos a explicar los criterios utilizados para determinar cuándo la población está evolucionando "demasiado" deprisa o despacio (criterios C_1 y C_2). Proponemos en este capítulo tres criterios diferentes para medir la velocidad de la búsqueda. Las medidas están basadas en el *fitness medio* (criterio AF), la *entropía de la población* (criterio PH), o en una combinación de los dos criterios previos (criterio AF+PH). Como estos criterios comprueban condiciones simples sobre el fitness medio y la entropía de la población (calculados en cada paso de la evolución en el paso de cálculo de estadísticas), podemos decir que son poco costosos de medir, a la vez que son indicativos del estado de la búsqueda. La complejidad para calcular el fitness medio es $O(\mu)$, mientras que en el caso de la entropía de la población es $O(\mu \cdot L)$, siendo μ el tamaño de la población y L la longitud de los cromosomas. Los detalles de estos criterios se dan a continuación:

- **AF**: Este criterio se basa en el *fitness medio* de la población. Por tanto, nuestra medida de la velocidad de convergencia se basa en términos de la diversidad del fenotipo. Definimos $\Delta\overline{f}_t$ como la diferencia entre los valores de fitness medio en las generaciones t y $t-1$: $\Delta\overline{f}_t = \overline{f}_t - \overline{f}_{t-1}$. El algoritmo cambiará el valor de la ratio al inmediatamente menor (manteniendo constante el tamaño de la población) si la diferencia $\Delta\overline{f}_t$ entre dos generaciones contiguas (t y $t-1$) se decrementa en al menos un factor de ε: $\Delta\overline{f}_t - \Delta\overline{f}_{t-1} < \varepsilon \cdot \Delta\overline{f}_{t-1}$ (condición C_1 del Algoritmo 6.1). Por otro lado, se cambiará la ratio al siguiente valor mayor si esa diferencia crece por encima de un factor $(1-\varepsilon)$: $\Delta\overline{f}_t - \Delta\overline{f}_{t-1} > (1-\varepsilon) \cdot \Delta\overline{f}_{t-1}$ (condición C_2). Formalmente, definimos C_1 y C_2 de la siguiente manera:

$$\begin{aligned} C_1 &::= \Delta\overline{f}_t < (1+\epsilon) \cdot \Delta\overline{f}_{t-1} \ , \\ C_2 &::= \Delta\overline{f}_t > (2-\epsilon) \cdot \Delta\overline{f}_{t-1} \ . \end{aligned}$$

- **PH**: Proponemos en este caso medir la velocidad de convergencia en términos de la diversidad del fenotipo. La *entropía de la población* es la métrica que utilizamos para este propósito. Calculamos esta entropía (H_t) como el valor medio de la entropía de cada gen en la población. Por tanto, este criterio es similar a AF excepto en que utiliza el cambio en la entropía de la población entre dos generaciones ($\Delta H_t = H_t - H_{t-1}$) en lugar de la variación del fitness medio. Consecuentemente, expresamos las condiciones C_1 y C_2 como:

$$\begin{aligned} C_1 &::= \Delta H_t < (1+\epsilon) \cdot \Delta H_{t-1} \ , \\ C_2 &::= \Delta H_t > (2-\epsilon) \cdot \Delta H_{t-1} \ . \end{aligned}$$

- **AF+PH**: Este es el tercer y último criterio propuesto para crear un cGA adaptativo. Este criterio considera tanto la entropía de la población como la aceleración en el valor del fitness medio mediante la combinación de los dos criterios anteriores (AF y PH). Por tanto, se consideran la diversidad de fenotipo y genotipo con el fin de obtener la mejor relación entre exploración y explotación. Éste es un criterio más restrictivo que los dos previos, ya que aunque la diversidad genética implica normalmente diversidad fenotípica, lo recíproco no es siempre verdad. La condición C_1 es en este caso el resultado de la operación lógica `and` de las condiciones C_1 de los dos criterios precedentes. De la misma forma, C_2 será la operación `and` de las condiciones C_2 de AF y PH:

$$\begin{aligned} C_1 &::= (\Delta\overline{f}_t < (1+\epsilon) \cdot \Delta\overline{f}_{t-1}) \ \textbf{and} \ (\Delta H_t < (1+\epsilon) \cdot \Delta H_{t-1}) \ , \\ C_2 &::= (\Delta\overline{f}_t > (2-\epsilon) \cdot \Delta\overline{f}_{t-1}) \ \textbf{and} \ (\Delta H_t > (2-\epsilon) \cdot \Delta H_{t-1}) \ . \end{aligned}$$

Además, para cada criterio adaptativo (AF, PH, y AF+PH) podemos comenzar la ejecución del algoritmo utilizando (i) la rejilla cuadrada, (ii) la más estrecha, o (iii) la que tenga un valor de la ratio intermedio (rectangular). En nuestra terminología, especificaremos la forma de rejilla inicial añadiendo al comienzo del nombre del criterio una s, n, o r para ejecuciones que comenzarán con las mallas cuadrada (Square), la más estrecha (Narrow), o una rectangular (la que tenga el valor medio de la ratio), respectivamente. Por ejemplo, al algoritmo que utilice el criterio AF comenzando con la población más estrecha se le llama nAF.

En resumen, hemos propuesto en esta sección tres criterios adaptativos diferentes. El primero, AF, se basa en la diversidad de fenotipo, mientras que el segundo, PH, se basa en la diversidad de genotipo. Finalmente, AF+PH es un criterio combinado que tiene en cuenta la diversidad a ambos niveles.

6.4. Experimentación

Presentamos en esta sección la comparación de los algoritmos presentados en la sección anterior. Para nuestras pruebas, hemos seleccionado un conjunto de problemas representativo, ya que contienen bastantes de las distintas características que podemos encontrar en optimización. En concreto hemos seleccionado problemas epistáticos, multimodales, engañosos, con restricciones, de identificación de parámetros, y generadores de problemas. Éstos son importantes ingredientes en cualquier trabajo que trate de evaluar aproximaciones algorítmicas con el objetivo de obtener resultados fiables, como se especifica en Whitley et al. [290].

Inicialmente, el Capítulo 5, ampliado con algunas instancias más de los problemas MAXCUT (instancias "cut20.01", "cut20.09" y "cut100"), y MTTP –instancias "mttp20", "mttp100" y "mttp200"– (consulte el Apéndice A para más información). La elección de esta batería de problemas se justifica tanto por la dificultad que entrañan los problemas que la componen como por los dominios de aplicación a los que pertenecen (optimización combinatoria, optimización continua, telecomunicaciones, planificación, etc.). Esto nos garantiza un elevado nivel de confidencia en los resultados, aunque la evaluación de las conclusiones será una tarea mucho más laboriosa que en el caso de utilizar un reducido número de problemas.

Aunque está fuera de nuestro objetivo realizar un estudio completo de los problemas seleccionados, en esta sección presentamos y analizamos los resultados obtenidos al resolver todos los problemas con las variantes de cGA antes presentadas (estáticas, pre-programadas y dinámicas), siempre utilizando el mismo vecindario (L5). Nótese que no es nuestro objetivo comparar el rendimiento de los cGAs con otros algoritmos y heurísticos pertenecientes al estado del arte para optimización combinatoria. Para ello, deberíamos al menos haber realizado un estudio previo para ajustar los parámetros del algoritmo, o incluir algún método de búsqueda local en el algoritmo, que no es el caso. Por tanto, los resultados únicamente pertenecen al rendimiento relativo entre los distintos cGAs propuestos.

El trabajo resultante es una continuación directa de un trabajo anterior [28], extendido aquí mediante la inclusión de un mayor número de problemas en nuestro banco de pruebas y algunos algoritmos nuevos (cGAs adaptativos). Además, realizamos las pruebas con dos esquemas de selección distintos: la selección por ruleta y la selección por torneo binario. Hemos elegido estos dos métodos de selección con el fin de analizar tanto técnicas de fitness proporcional como de *ranking*. Por un lado, la selección por ruleta nos permitirá probar hasta qué punto son importantes en los algoritmos propuestos los errores en la selección. Por otro lado, usando la selección por torneo deberíamos obtener resultados similares a los obtenidos con otros métodos de *ranking*, pero hemos seleccionado torneo en lugar de *ranking* debido a su menor coste computacional. Además, la selección por torneo se utiliza para comprobar si muestra algún tipo de mejora en la búsqueda con respecto a la ruleta. En resumen, incluyendo estos dos métodos de selección (y no sólo uno de ellos) pretendemos ofrecer un análisis más completo.

Procedemos a realizar nuestro estudio en tres etapas, en concreto, el análisis de las ratios estáticas, el de las ratios pre-programadas, y el de las ratios dinámicas adaptativas. En los dos primeros pasos utilizamos mallas cuadradas y rectangulares en puntos bien conocidos *a priori* de la ejecución, es decir, bien utilizando siempre la misma malla (estática), o estableciendo de antemano un cambio en la forma de la rejilla antes de la optimización.

En la tercera etapa aplicamos tres mecanismos (dinámicos) de adaptación de la malla basados en la velocidad de convergencia, como se explicó en la Sección 6.3.2. Estudiamos estos mecanismos con dos valores iniciales de la ratio diferentes: el menor, y el valor medio (es decir, la mediana del conjunto de valores permitidos para la ratio).

Todos los algoritmos han sido comparados en términos de eficiencia y eficacia, así que las tablas 6.3, 6.4, 6.5, y 6.6 muestran el número medio de evaluaciones necesario para resolver cada uno de los problemas con todos los algoritmos y el porcentaje de ejecuciones con éxito tras realizar 100 ejecuciones independientes para cada problema. Se han aplicado análisis estadísticos sobre los resultados con el fin de obtener conclusiones estadísticamente significativas en nuestras comparaciones (vea las tablas 6.3 a 6.6).

Esta sección está organizada en cinco sub-secciones. En la primera detallamos los parámetros utilizados en la ejecución de los algoritmos. En la siguiente compararemos el comportamiento de un cGA y un GA con población panmíctica. En las dos secciones siguientes presentamos y analizamos los resultados obtenidos con los distintos algoritmos para los dos métodos de selección propuestos. Finalmente, ofrecemos en la última sección una interpretación gráfica global adicional de los resultados obtenidos.

6.4.1. Parametrización

Para hacer una comparación significativa entre los algoritmos hemos utilizado una parametrización común. Los detalles se encuentran en la Tabla 6.1, donde L es la longitud de la cadena que representa el cromosoma de los individuos. Hemos utilizado dos esquemas de selección de individuos diferentes en este trabajo. Uno de los padres es siempre el individuo actual, mientras que el otro se selecciona utilizando *torneo binario* –BT– (Sección 6.4.3) o *torneo por ruleta* –RW– (Sección 6.4.4). Se obliga a que los dos padres sean distintos individuos.

Tabla 6.1: Parametrización utilizada en los algoritmos.

Tamaño de Población	400 individuos
Selección de Padres	actual + (BT o RW)
Recombinación	DPX, $p_c = 1.0$
Mutación de Bit	Bit-flip, $p_m = 1/L$
Reemplazo	Reemp_si_Mejor

En el operador de recombinación obtenemos un único descendiente de los dos padres: aquel con la porción más larga del mejor padre. El operador de recombinación *DPX* se aplica siempre, con probabilidad $p_c = 1.0$. La probabilidad de mutación de bit es $p_m = 1/L$, con las excepciones de los problemas COUNTSAT, donde usamos $p_m = 1/(2 \cdot L)$, y FMS, para el que usamos $p_m = 10/L$. El uso de estos dos valores de probabilidad distintos es necesario debido a que los algoritmos obtienen una tasa de éxito insignificante con la probabilidad estándar $p_m = 1/L$ para estos dos problemas.

El individuo considerado será reemplazado en cada generación sólo si su descendiente tiene un mejor valor de fitness. Esta estrategia de reemplazo se llama *reemplaza si es mejor* [256]. El coste de resolver un problema se calcula midiendo el número de evaluaciones de la función de fitness realizado durante la búsqueda. Finalmente, el criterio de parada para todos los algoritmos es encontrar una solución al problema o alcanzar un número máximo de evaluaciones de función.

Como ya introdujimos en el Capítulo 4, hemos realizado un análisis estadístico sobre los resultados medios para poder asegurar que los resultados sean estadísticamente significativos, y consideramos un nivel significativo de 0.05. La existencia de diferencias estadísticamente significativas entre todos los algoritmos se muestra con símbolos '+' en las tablas 6.3 a 6.6, mientras que los casos en los que no se encuentre confianza estadística en los resultados serán indicados con el símbolo '•'.

En resumen, tenemos once algoritmos diferentes utilizando la selección RW y otros once con BT. El resto de los operadores de variación son los mismos para los 22 algoritmos (consulte la Tabla 6.1).

Para cada método de selección, tres algoritmos utilizan ratios estáticas, otros dos usan ratios dinámicas pre-programadas, y los seis restantes utilizan los criterios dinámicos adaptativos propuestos.

6.4.2. Estudiando los Efectos de Estructurar la Población

Con el fin de justificar el uso de una población descentralizada, presentamos en esta sección los resultados obtenidos por dos GAs generacionales trabajando sobre poblaciones panmícticas, y dos cGAs con población cuadrada. Todos estos algoritmos trabajan con los parámetros presentados en la sección anterior.

Mostramos los resultados en la Tabla 6.2 (los mejores resultados están resaltados en **negrita**). Como se puede ver, los GAs con poblaciones estructuradas mejoran a aquellos con población panmíctica (no estructurada) en casi todos los problemas. La tasa de éxito de los algoritmos es siempre más alta (o igual en algunos casos) para los cGAs con respecto al caso de los correspondientes algoritmos no estructurados. Además, de entre todos los algoritmos estudiados en este capítulo, estos dos algoritmos con población panmíctica son los únicos que obtienen la indeseable tasa de éxito del 0 % para algunos problemas. La tasa de éxito media para todos los problemas es considerablemente más elevada en el caso de los cGAs para los dos métodos de selección estudiados, con un valor del 85.36 % para BT y un 87.46 % en el caso de RW, con respecto a los algoritmos panmícticos (con valores del 54.72 % para BT y el 46.64 % para RW).

En términos del número medio de evaluaciones, la Tabla 6.2 muestra que, en general, los algoritmos celulares son también más rápidos que los que utilizan la población panmíctica. Además, queremos destacar las importantes diferencias en el comportamiento de los algoritmos panmícticos dependiendo del método de selección utilizado: parece ser bastante importante para la mayoría de los problemas estudiados.

6.4.3. Resultados Experimentales Utilizando Torneo Binario

En esta sección presentamos los resultados obtenidos al resolver con todos los algoritmos presentados anteriormente los problemas seleccionados para nuestro estudio. El método de selección utilizado en todos los algoritmos aquí tratados es el torneo binario. Utilizando esta selección se espera que se reduzca el error de muestreo en vecindarios pequeños con respecto a una selección proporcional al valor de fitness. Los mejores valores obtenidos en todo este capítulo para cada problema con este método de selección se resaltan en **negrita**; por ejemplo, el algoritmo que resuelve el problema FMS con un menor esfuerzo de entre los probados –utilizando BT– es *SqNar* (criterio pre-programado), como se muestra en la Tabla 6.3.

Organizamos el resto de esta sección en dos partes. En la primera estudiamos los resultados obtenidos por los diferentes criterios estáticos y pre-programados para todos los problemas. En la segunda parte realizamos el mismo estudio, pero en este caso sobre los algoritmos adaptativos.

Ratios Estáticas y Pre-programadas

En esta sección consideramos la optimización de todos los problemas utilizando ratios estáticas y pre-programadas con el operador de selección BT. Nuestros resultados confirman los publicados en [28], en los que las ratios estrechas eran bastante eficientes en problemas multimodales y/o engañosos, mientras que las rejillas cuadrada y rectangular se comportaban mejor en el caso de los problemas más simples y no engañosos. Debe tenerse en cuenta que en este caso estamos utilizando el método de selección BT, mientras que en [28] se utilizó RW.

Observando la Tabla 6.3 podemos ver que el uso de rejillas estrechas es más apropiado para problemas multimodales y engañosos (Narrow obtiene mejores resultados que los demás para P-PEAKS, y SqNar obtiene los mejores resultados para FMS). En contraste, estas rejillas estrechas se comportan peor que los

Tabla 6.2: Resultados (número medio de evaluaciones y porcentaje de éxito) con poblaciones no estructurada (panmíctica) y estructurada (celular con rejilla de 20×20 individuos).

Problema	**Población Panmíctica**		**Población Celular**	
	Torneo Binario	**Ruleta**	**Torneo Binario**	**Ruleta**
MMDP	555505.4	—	160090.30	**157167.05**
	93 %	0 %	**93** %	89 %
FMS	—	—	**435697.94**	520537.23
	0 %	0 %	59 %	**87** %
P-PEAKS	**20904.0**	703250.0	54907.93	51499.43
	100 %	8 %	**100** %	**100** %
COUNTSAT	—	—	**6845.56**	6846.32
	0 %	0 %	24 %	**28** %
ECC	**47887.0**	754133.3	167701.21	153497.79
	46 %	6 %	**100** %	**100** %
"cut20.01"	16052.0	11548.0	6054.10	**5103.73**
	100 %	**100** %	**100** %	**100** %
"cut20.09"	14700.0	85940.0	**9282.15**	9542.80
	100 %	**100** %	**100** %	**100** %
"cut100"	319520.0	—	263616.77	**176591.80**
	5 %	0 %	**63** %	58 %
"mttp20"	**4584.0**	4860.0	6058.11	5496.71
	100 %	**100** %	**100** %	**100** %
"mttp100"	467901.8	48412.0	194656.43	**177950.77**
	57 %	**100** %	**100** %	**100** %
"mttp200"	962400.0	201200.0	565364.89	**514706.56**
	1 %	99 %	**100** %	**100** %

Tabla 6.3: Resultados con ratios estáticas y pre-programadas utilizando selección por torneo binario.

Problema	**Estático**			**Pre-programado**		**Test**
	Square	**Rectangular**	**Narrow**	**SqNar**	**NarSq**	
MMDP	160090.30	182305.67	247841.13	148505.48	172509.28	+
	93 %	**96 %**	93 %	86 %	92 %	
FMS	435697.94	494400.25	499031.79	**355209.39**	462288.25	+
	59 %	67 %	68 %	42 %	59 %	
P-PEAKS	54907.93	54984.12	50853.82	63545.47	60674.31	+
	100 %	**100 %**	**100 %**	**100 %**	**100 %**	
COUNTSAT	6845.56	6851.26	6856.77	6845.18	6849.17	•
	24 %	54 %	83 %	22 %	43 %	
ECC	167701.21	169469.62	156541.38	190875.00	187658.98	+
	100 %	**100 %**	**100 %**	**100 %**	**100 %**	
"cut20.01"	6054.10	5769.39	5713.25	5929.79	5785.43	•
	100 %	**100 %**	**100 %**	**100 %**	**100 %**	
"cut20.09"	**9282.15**	9606.96	9899.69	9775.38	10260.59	•
	100 %	**100 %**	**100 %**	**100 %**	**100 %**	
"cut100"	263616.77	217509.87	197977.18	223909.83	240346.84	•
	63 %	55 %	53 %	55 %	53 %	
"mttp20"	6058.11	5753.35	5681.17	6134.30	**5584.93**	+
	100 %	**100 %**	**100 %**	**100 %**	**100 %**	
"mttp100"	194656.43	201409.27	206177.16	190566.23	194371.72	+
	100 %	**100 %**	**100 %**	**100 %**	**100 %**	
"mttp200"	565364.89	566279.17	606146.59	565465.14	570016.49	+
	100 %	**100 %**	**100 %**	**100 %**	**100 %**	

demás en el caso de MMDP (engañoso), y para algunas instancias de MTTP (optimización combinatoria). Finalmente, los dos criterios pre-programados obtienen los peores rendimientos en el caso de ECC. En los demás problemas estudiados no existen diferencias significativas estadísticamente.

Aunque no se trata de una estrategia globalmente válida para cualquier problema no estudiado aquí, nuestras conclusiones hasta el momento explican la creencia común de que algoritmos de optimización "buenos" deben buscar inicialmente regiones prometedoras y después ir buscando gradualmente en el vecindario de los mejores puntos. En este proceso, existe una clara necesidad de evitar una convergencia total hacia un óptimo local, lo que describe exactamente el comportamiento de SqNar, ya que cambia a una rejilla estrecha en la segunda parte de la búsqueda para evitar esta prematura convergencia. En efecto, este algoritmo tiene una gran precisión y eficiencia en el conjunto de problemas estudiado.

En resumen, estos resultados confirman también el teorema de No Free Lunch (NFL) [294], que predice que no es posible encontrar un algoritmo "globalmente mejor" para el conjunto de todos los posibles problemas. Por tanto, hemos incluido en nuestro conjunto de problemas de pruebas un elevado número de problemas de diferentes características, ya que analizando únicamente dos o tres problemas podríamos caer en prejuicios no deseables en nuestras conclusiones. Cada algoritmo ha destacado como el más eficiente para una subclase diferente de problemas. En cualquier caso, si inspeccionamos la Tabla 6.3, podemos observar que sólo hay tres resultados en negrita (con diferencias estadísticamente significativas para FMS, en general), lo que significa que el algoritmo más eficiente globalmente está aún por descubrirse en este trabajo. Como veremos, los diferentes criterios adaptativos evaluados en la siguiente sección serán capaces de encontrar la solución óptima con un menor coste con respecto a todas las ratios no adaptativas para todos los problemas estudiados (con la excepción del problema FMS).

Criterios Adaptativos

En esta sección continuamos nuestro trabajo llevando a cabo el estudio de los algoritmos adaptativos propuestos en la Sección 6.3.2. Para cada criterio adaptativo, se han realizado dos pruebas diferentes: una comenzando con la rejilla teniendo un valor medio de la ratio (r), y la otra comenzando con la rejilla de menor ratio (n).

Además, como los criterios adaptativos se apoyan en pequeños valores ε para definir qué se entiende por "convergencia demasiado rápida", hemos llevado a cabo un primer conjunto de pruebas para todos los criterios con diferentes valores de ε: 0.05, 0.15, 0.25 y 0.3. De esta plétora de experimentos hemos seleccionado finalmente el valor $\varepsilon = 0.05$ como el mejor. Para llegar a esta conclusión hemos aplicado dos criterios distintos, y en ambos $\varepsilon = 0.05$ fue el mejor. Por supuesto, otros valores distintos de $\varepsilon = 0.05$ podrían reducir el esfuerzo para algún problema dado, pero necesitamos seleccionar un valor determinado para enfocar nuestra discusión. Los criterios de selección empleados están enfocados en buscar un valor de ε que nos permita obtener (i) la mejor precisión, y (ii) la mejor eficiencia. Queríamos hacer un estudio formal y estadístico como el realizado para evitar el tener que hacer una definición *ad hoc* de ε, pero no mostramos los detalles en este documento de tesis, ya que incluir todos los resultados, tablas, y explicaciones podría distraer la atención del lector. El lector interesado en dichos detalles puede consultarlos en [15].

A continuación discutimos los resultados obtenidos con nuestros cGAs adaptativos utilizando la selección BT (mostrados en la Tabla 6.4), y también compararemos para todos los problemas nuestro mejor algoritmo adaptativo con todos los algoritmos estáticos y pre-programados. Sólo en unos pocos casos existen diferencias estadísticamente significativas entre los criterios adaptativos de la Tabla 6.4. Simplemente destacaremos el comportamiento altamente deseable de nPH y rPH, que son (con significancia estadística) los mejores criterios adaptativos para P-PEAKS y ECC (y un poco peores que rAF

Tabla 6.4: Resultados con los criterios adaptativos utilizando selección por torneo binario.

Problema	***n*AF**	***r*AF**	***n*PH**	***r*PH**	***n*AF+PH**	***r*AF+PH**	**Test**
MMDP	155342.46	**137674.50**	180601.45	160747.96	144483.41	162039.13	+
	93 %	83 %	93 %	91 %	90 %	**96 %**	
FMS	493449.91	491922.13	528850.10	555155.67	465396.00	439310.97	+
	64 %	61 %	73 %	**76 %**	59 %	57 %	
P-PEAKS	56475.84	58031.72	**49253.83**	49855.33	53127.49	54350.54	+
	100 %	**100 %**	**100 %**	**100 %**	**100 %**	**100 %**	
COUNTSAT	5023.71	**4678.77**	5296.32	5188.08	4999.59	5252.22	•
	82 %	**90 %**	89 %	86 %	88 %	88 %	
ECC	178921.19	179081.59	**151256.20**	156008.05	193373.23	184058.00	+
	100 %	**100 %**	**100 %**	**100 %**	**100 %**	**100 %**	
"cut20.01"	5416.51	5641.07	5701.22	6082.17	**5340.32**	5657.11	•
	100 %	**100 %**	**100 %**	**100 %**	**100 %**	**100 %**	
"cut20.09"	9554.83	9659.09	9430.52	10015.98	9935.78	9598.94	•
	100 %	**100 %**	**100 %**	**100 %**	**100 %**	**100 %**	
"cut100"	131359.13	142470.83	151048.22	**111662.07**	132574.18	141945.57	•
	45 %	46 %	46 %	39 %	43 %	41 %	
"mttp20"	5729.29	5941.82	6206.48	5941.82	6038.06	5853.60	•
	100 %	**100 %**	**100 %**	**100 %**	**100 %**	**100 %**	
"mttp100"	190425.88	192179.23	**190081.02**	196958.17	192871.98	192579.25	•
	100 %	**100 %**	**100 %**	**100 %**	**100 %**	**100 %**	
"mttp200"	529158.60	523913.52	**523372.17**	525349.10	531019.24	541397.12	•
	100 %	**100 %**	**100 %**	**100 %**	**100 %**	**100 %**	

y nAF+PH para el problema FMS).

Si comparamos los criterios adaptativos con los cGAs que utilizan ratios estáticas y pre-programadas, podemos ver (tablas 6.3 y 6.4) que los algoritmos adaptativos son más eficientes en general que los estáticos y pre-programados, destacando el algoritmo nPH sobre los demás.

En términos de la eficiencia de los algoritmos (tasa de éxito), podemos ver que, en general, los algoritmos adaptativos encuentran la solución óptima más frecuentemente en las 100 ejecuciones realizadas que los estáticos y pre-programados. De hecho, estos algoritmos adaptativos obtienen las mejores tasas de éxito para prácticamente todos los problemas.

6.4.4. Resultados Experimentales Utilizando Ruleta

En esta sección extenderemos el estudio previo utilizando en este caso la selección por ruleta (éste es el método utilizado por Alba et al. en [28]) en lugar de BT. Comenzaremos explicando los valores obtenidos por los cGAs utilizando ratios estáticas, pre-programadas y adaptativas. Todos estos resultados se muestran en las tablas 6.5 y 6.6. Como en la Sección 6.4.3, las cifras de estas tablas se corresponden con el número medio de evaluaciones de función necesarios para encontrar el óptimo, la tasa de éxito, y la confidencia estadística de la comparación de los algoritmos. Los mejores valores entre todos los criterios para cada problema están en **negrita**.

La sección está estructurada de una forma similar a la sección previa. Por tanto, en la primera parte estudiamos los algoritmos estáticos y pre-programados, mientras que en la segunda analizamos los criterios adaptativos.

Tabla 6.5: Resultados con ratios estáticas y pre-programadas utilizando selección por ruleta.

Problema	**Estático**			**Pre-programado**		**Test**
	Square	**Rectangular**	**Narrow**	**SqNar**	**NarSq**	
MMDP	157167.05	183717.22	248984.00	173279.26	162893.30	+
	89 %	93 %	91 %	86 %	92 %	
FMS	520537.23	545591.67	481716.45	525032.49	552079.84	•
	87 %	91 %	84 %	82 %	92 %	
P-PEAKS	51499.43	50837.78	**47325.02**	57594.63	58176.08	+
	100 %	**100 %**	**100 %**	**100 %**	**100 %**	
COUNTSAT	6846.32	6850.50	6857.91	6846.89	6851.07	•
	28 %	50 %	89 %	31 %	53 %	
ECC	153497.79	151444.67	**148393.06**	168551.33	164100.23	+
	100 %	**100 %**	**100 %**	**100 %**	**100 %**	
"cut20.01"	5103.73	5224.03	**5075.66**	5869.64	5424.53	+
	100 %	**100 %**	**100 %**	**100 %**	**100 %**	
"cut20.09"	9542.80	9294.18	9406.46	9923.75	9522.75	•
	100 %	**100 %**	**100 %**	**100 %**	**100 %**	
"cut100"	176591.80	154500.80	181969.32	184756.24	171190.46	•
	58 %	49 %	47 %	50 %	45 %	
"mttp20"	5496.71	5653.10	**5236.06**	5260.12	5532.80	•
	100 %	**100 %**	**100 %**	**100 %**	**100 %**	
"mttp100"	177950.77	169690.17	203446.35	168334.79	177746.26	+
	100 %	**100 %**	**100 %**	**100 %**	**100 %**	
"mttp200"	514706.56	533818.22	588835.42	525176.67	531195.68	+
	100 %	**100 %**	**100 %**	**100 %**	**100 %**	

Ratios Estáticas y Pre-programadas

Prestemos atención en primer lugar a los criterios estáticos. Nuestra primera conclusión es que nuestros resultados confirman los desprendidos del estudio realizado en [28], como sucede en el caso de los algoritmos utilizando BT. En concordancia con este trabajo previo, concluimos que la ratio estática Narrow es más eficiente en el caso de problemas altamente engañosos y multimodales (FMS y P-PEAKS), donde su reducida presión de selección promociona la exploración del algoritmo; aunque la diferencia es significativa sólo para P-PEAKS. Por otro lado, la elevada presión de selección de la ratio estática Square parece ser más apropiada para el caso de una función engañosa y multimodal como MMDP. Además, también concluimos que la ratio Square se comporta mejor que las otras dos ratios estáticas para problemas combinatorios como MAXCUT o MTTP (para éste último, Narrow es el peor criterio con confianza estadística).

Con respecto a los criterios pre-programados, el lector puede verificar (tras consultar la Tabla 6.5) que la ratio NarSq (que representa el cambio de exploración a explotación) no mejora los mejores resultados obtenidos con respecto al resto de los criterios estudiados con ratios estáticas y pre-programadas en ningún caso. Además, el otro criterio pre-programado, SqNar, (que representa el cambio hacia una mayor diversidad en la segunda fase del algoritmo) únicamente mejora en eficiencia a los demás criterios de la Tabla 6.5 en el caso de la instancia "mttp100", aunque no existe diferencia estadísticamente significativa. Los dos criterios pre-programados son peores que todos los estáticos en los casos de los problemas ECC y P-PEAKS con confianza estadística.

Tabla 6.6: Resultados con los criterios adaptativos utilizando selección por ruleta.

Problema	***n*AF**	***r*AF**	***n*PH**	***r*PH**	***n*AF+PH**	***r*AF+PH**	**Test**
MMDP	161934.96	153830.66	190365.78	168471.30	**142911.49**	172794.01	+
	87 %	**96 %**	95 %	83 %	90 %	90 %	
FMS	503863.63	529495.51	512789.91	**478175.61**	515448.56	538109.07	+
	89 %	**93 %**	87 %	85 %	85 %	85 %	
P-PEAKS	54390.64	54222.22	49518.49	48985.16	51423.24	51327.00	+
	100 %	**100 %**	**100 %**	**100 %**	**100 %**	**100 %**	
COUNTSAT	6858.86	5284.26	5292.34	**4947.50**	5284.33	5725.35	•
	94 %	92 %	86 %	84 %	85 %	93 %	
ECC	157239.12	160840.10	151344.42	148425.14	166794.95	168535.29	+
	100 %	**100 %**	**100 %**	**100 %**	**100 %**	**100 %**	
"cut20.01"	5452.60	5705.23	5937.81	5436.56	5484.68	5436.56	•
	100 %	**100 %**	**100 %**	**100 %**	**100 %**	**100 %**	
"cut20.09"	**8953.33**	9322.25	9803.45	9117.74	9326.26	9338.29	•
	100 %	**100 %**	**100 %**	**100 %**	**100 %**	**100 %**	
"cut100"	**130035.85**	178652.97	140662.35	193774.65	157464.19	143617.70	•
	43 %	55 %	43 %	**58 %**	49 %	45 %	
"mttp20"	5520.77	5665.13	5781.42	5584.93	5572.90	5889.69	•
	100 %	**100 %**	**100 %**	**100 %**	**100 %**	**100 %**	
"mttp100"	170861.09	**166325.78**	192338.65	197258.92	180244.49	180757.77	+
	100 %	**100 %**	**100 %**	**100 %**	**100 %**	**100 %**	
"mttp200"	**494022.98**	495542.77	527261.87	524799.73	508446.95	499215.93	+
	100 %	**100 %**	**100 %**	**100 %**	**100 %**	**100 %**	

Criterios Adaptativos

Como hicimos previamente en la Sección 6.4.3, procederemos a estudiar el comportamiento de nuestros algoritmos adaptativos con el método de selección RW, y los compararemos con los algoritmos que utilizan las ratios estáticas y pre-programadas. En la Tabla 6.6 mostramos los resultados obtenidos con los seis criterios adaptativos. La primera conclusión que se desprende de esta tabla es que, como en el caso de la Sección 6.4.3, el comportamiento de los cGAs se mejora globalmente cuando los hacemos adaptativos (nótese el alto número de resultados marcados en negrita).

Además, los algoritmos más efectivos para cada problema son siempre los adaptativos. El lector puede verificar en las tablas 6.5 y 6.6 que las tasas de éxito más altas para todos los problemas pertenecen también a los algoritmos adaptativos. Por tanto, concluimos que, en general, los algoritmos adaptativos (bien usando torneo binario o ruleta) mejoran los otros estudiados en los dos aspectos considerados: eficiencia y eficacia.

En la Tabla 6.7 comparamos, para cada problema, nuestro mejor algoritmo adaptativo con los algoritmos estáticos y pre-programados. Esta tabla tiene en cuenta los algoritmos utilizando tanto el método de selección BT como RW. El símbolo '+' significa que el algoritmo adaptativo es mejor con confianza estadística que los no adaptativos comparados, mientras que '•' se emplea para denotar que no existen diferencias significativas. Podemos ver que, para los dos métodos de selección, no existe ningún algoritmo estático o pre-programado mejor que el mejor adaptativo para cada problema. Por tanto, los valores resaltados en negrita en las tablas 6.3 y 6.5 correspondientes a un algoritmo no adaptativo mejor son indistinguibles de los adaptativos. Para todos los problemas, las diferencias estadísticamente significativas existentes favorecen a los criterios adaptativos frente a los estáticos y pre-programados.

Tabla 6.7: Comparación del mejor criterio adaptativo con las ratios no adaptativas utilizando selección por torneo binario y por ruleta (BT,RW).

Problema	**Mejor Criterio Adaptativo (BT,RW)**			**Square**	**Rectangular**	**Narrow**	**SqNar**	**NarSq**
MMDP	rAF	,	nAF+PH	•,•	•,+	+,+	•,+	•,•
FMS	rAF+PH	,	rPH	•,•	•,•	•,•	•,+	•,•
P-PEAKS	nPH	,	rPH	+,+	+,•	•,•	+,+	+,+
COUNTSAT	rAF	,	rPH	+,•	•,•	•,•	•,•	•,•
ECC	nPH	,	rPH	+,•	+,•	•,•	+,+	+,+
"cut20.01"	nAF+PH	,	rPH	•,•	•,•	•,•	•,•	•,•
"cut20.09"	nPH	,	nAF	•,•	•,•	•,•	•,•	•,•
"cut100"	rPH	,	nAF	•,•	•,•	•,•	•,•	+,•
"mttp20"	nAF	,	nAF	•,•	•,•	•,•	•,•	•,•
"mttp100"	nPH	,	rAF	•,•	•,•	+,+	•,•	•,•
"mttp200"	nPH	,	nAF	+,•	+,+	+,+	+,+	+,+

6.4.5. Discusión Adicional

Si buscamos entre todos los criterios adaptativos podemos concluir (como se esperaba de la teoría) que no existe un criterio no adaptativo que mejore significativamente al resto en todo nuestro conjunto de problemas, de acuerdo a los test estadísticos realizados. Por tanto, es claro tras estos resultados que las ratios adaptativas son el primer tipo de algoritmo que se debe seleccionar con el fin de tener un buen punto de partida para las comparaciones. Para los problemas evaluados, y para otros problemas que compartan las mismas características, hacemos una contribución definiendo nuevos algoritmos competitivos y robustos que pueden mejorar el rendimiento existente a costa de un mínimo requerimiento de implementación (el cambio de la forma de la rejilla).

Con el fin de poder facilitar una explicación intuitiva de qué está sucediendo durante la búsqueda que realizan estos algoritmos adaptativos presentamos en primer lugar un seguimiento de la evolución del valor de la ratio durante la búsqueda. En segundo lugar, hemos tomado algunas fotos de la población en diferentes momentos de la evolución con el fin de estudiar la difusión de las soluciones a través de la población. Finalmente, hemos estudiado un escenario de la convergencia representativa de un algoritmo adaptativo a lo largo de la evolución.

En la Figura 6.6 mostramos un ejemplo de la fluctuación automática de la ratio producida por el criterio adaptativo rAF durante una ejecución típica de todos los problemas (mostramos únicamente las instancias más duras en el caso de los problemas MAXCUT y MTTP: "cut100" y "mttp200"). Se encuentran separados en la gráfica de la derecha los problemas más duros: MTTP con 200 variables –"mttp200"– y FMS. Se ha seleccionado el criterio rAF para describir los escenarios típicamente encontrados en todos los algoritmos adaptativos.

Podemos apreciar en la Figura 6.6 una clara tendencia hacia la promoción de la explotación de la población (ratio mayor que 0.1). Podríamos haber esperado esta tendencia general a evolucionar hacia la rejilla cuadrada, ya que el proceso de búsqueda se enfoca en la explotación de la población tras una primera fase de exploración.

Este cambio de ratios pequeñas a elevadas se detecta también para todos los problemas (gráfica de la izquierda en la Figura 6.6). En cualquier caso, los algoritmos adaptativos parecen necesitar una búsqueda más compleja de la ratio óptima cuando se afrontan los problemas más complejos: FMS y MTTP. En estos dos casos (gráfica de la derecha en la Figura 6.6), existen breves y periódicas visitas a zonas correspondientes a valores de ratios menores (bajo 0.08) con el objetivo de mejorar la exploración del algoritmo. Este cambio alternado automático entre explotación y exploración para algunos problemas es una característica destacable del mecanismo adaptativo que proponemos, ya que muestra cómo el algoritmo decide él solo cuándo explorar y cuándo explotar la población.

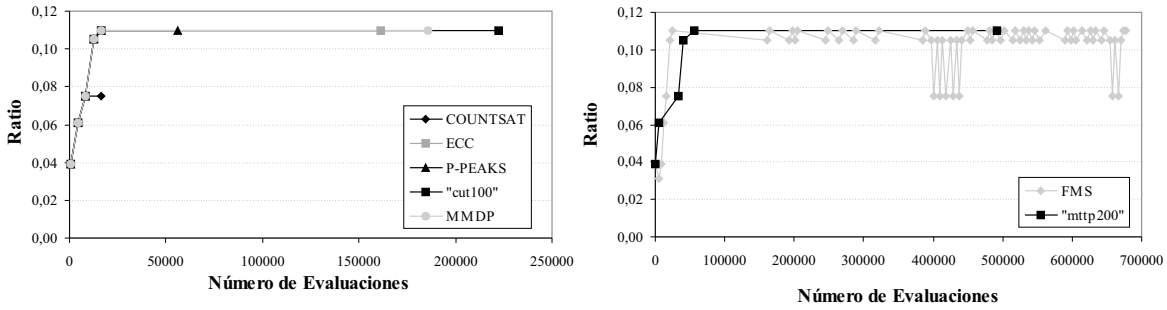


Figura 6.6: Dinámica de la ratio observada al utilizar rAF.

Pero queremos mostrar que esta búsqueda del equilibrio apropiado entre exploración y explotación puede ser alcanzado de una manera gradual y progresiva. Con el fin de probar esta afirmación, nos centramos en el comportamiento del algoritmo nAF+PH para el problema FMS (Figura 6.7). Esto nos permite darnos cuenta de que cuando el algoritmo encuentra dificultades durante una fase de explotación (ratio elevada), disminuye el valor de la ratio automáticamente (incluso repetidamente) para mejorar la exploración. Esto nos proporciona, en general, una tasa de éxito mayor para los cGAs adaptativos con respecto a los otros cGAs estudiados (estáticos y pre-programados). En la Figura 6.7 podemos ver que la caída del valor de la ratio se realiza por el algoritmo adaptativo con el fin de regular apropiadamente la aceleración de la búsqueda del óptimo de forma automática.

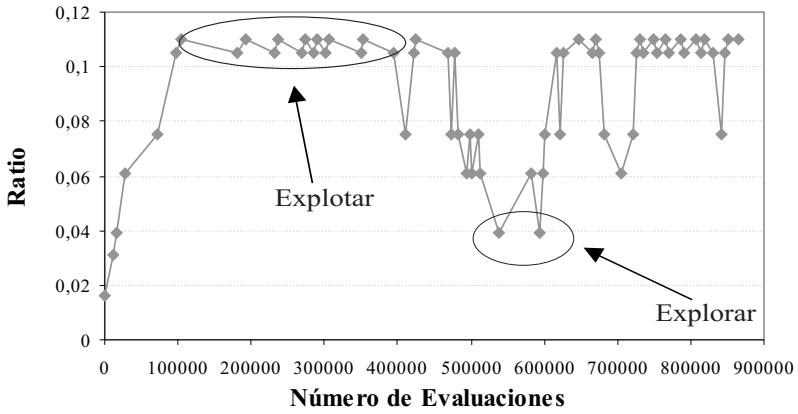


Figura 6.7: Evolución típica del valor de la ratio para el problema FMS con el algoritmo nAF+PH.

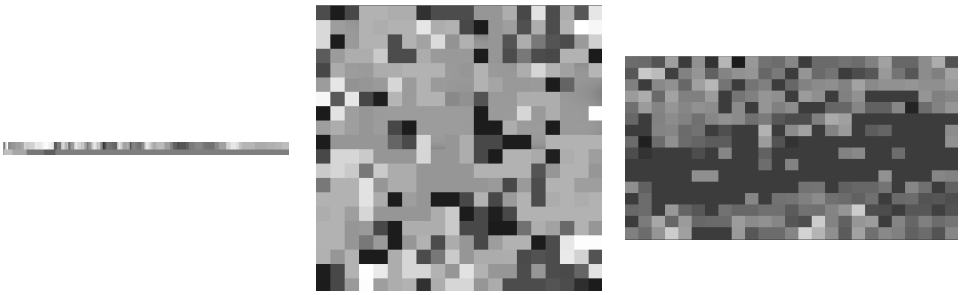
Ahora, analicemos la evolución para MAXCUT y FMS desde el punto de vista de la diversidad fenotípica. Las imágenes de las figuras 6.8 y 6.9 han sido tomadas al comienzo, en la mitad, y al final de una ejecución típica con un criterio adaptativo (concretamente nAF+PH) que comienza utilizando la forma más estrecha permitida para la población. Estas figuras muestran tres representaciones fotográficas de la distribución del fitness en la población en diferentes etapas de la búsqueda. Los tonos de gris diferentes representan distintos valores de fitness, y los tonos más oscuros se corresponden con los mejores individuos. Se puede ver en las dos figuras cómo la diversidad (variedad de los tonos de gris) decrece durante la ejecución, como era de esperar.

(a) Rejilla: 2 x 200 (b) Rejilla: 20 x 20 (c) Rejilla: 20 x 20

Figura 6.8: Población al comienzo (a), mitad (b), y final (c) de un ejemplo de ejecución para MAXCUT (instancia "cut100") con nAF+PH.

El caso particular de MAXCUT con la instancia "cut100" (Figura 6.8) es un ejemplo de escenario de una evolución rápida hacia la población cuadrada durante la ejecución del algoritmo; una vez que el algoritmo ha alcanzado la población cuadrada, se mantiene hasta que la solución se encuentre (compruebe este resultado con el mostrado en la Figura 6.6). La razón de este comportamiento es que el equilibrio entre exploración y explotación que encuentra el algoritmo es adecuado para mantener una convergencia permanente y no demasiado rápida de los individuos hacia la solución al problema. Esta convergencia se puede observar en la Figura 6.8, donde podemos ver cómo el número de diferentes tonos de gris en la población disminuye durante la ejecución. La convergencia es apreciable en la mitad de la ejecución (Figura 6.8b) y continúa creciendo hasta que se encuentra la solución final (Figura 6.8c). Además, como se mantiene durante toda la evolución el mismo valor de la ratio (con la excepción de las primeras generaciones), no se produce la relocalización de los individuos (que introduce más diversidad), y el algoritmo crea de forma inherente zonas de individuos con valores de fitness similares, también llamados *nichos* (Figura 6.8c), correspondientes a aquellas zonas coloreadas con el mismo tono de gris. Estas áreas diferentes representan distintos caminos de convergencia, y están motivados por la suave difusión de las mejores soluciones, una característica distintiva de los cGAs. En particular, la exploración del espacio de búsqueda se encuentra especialmente impuesta en los bordes de estos nichos.

En el caso del problema FMS, nos encontramos con un escenario similar; en este caso mostramos una situación de ejemplo en la que el algoritmo se encuentra cerca de converger hacia un óptimo local. En cuanto el criterio adaptativo detecta dicha condición, intenta salir de dicho óptimo local promoviendo la exploración (como ocurre también en la Figura 6.7). Podemos ver en la Figura 6.9b cómo la diversidad es menor que en el caso de MAXCUT en la mitad de la ejecución. Como la mayoría de los individuos tienen valores de fitness parecidos, cuando aparece una solución mejor en la población se esparcirá por toda la población de manera razonablemente rápida. Este efecto produce una aceleración en el fitness medio de los individuos y, por tanto, se realiza un cambio a la siguiente rejilla más estrecha (si es posible), introduciendo de esta forma más diversidad en los vecindarios para las siguientes generaciones. En contraposición a este razonamiento, se mantiene durante la evolución un elevado nivel de diversidad ya que, debido a la naturaleza engañosa del problema, pequeños cambios en los individuos llevan usualmente a grandes diferencias en los valores de fitness. Ésta es la razón, además del alto número de cambios de la ratio realizados para este problema, de la ausencia de áreas uniformemente coloreadas. Esto también justifica los niveles importantes de diversidad presentes al final de la evolución para FMS (en relación con el problema MAXCUT).



(a) Rejilla: 2 x 200 (b) Rejilla: 20 x 20 (c) Rejilla: 16 x 25

Figura 6.9: Población al comienzo (a), mitad (b), y final (c) de un ejemplo de ejecución para FMS con nAF+PH.

Finalmente, queremos ilustrar las diferencias en la velocidad de convergencia de las tres principales clases de algoritmos estudiados (estático, pre-programado y adaptativo). Para ello, mostramos en la Figura 6.10 el valor de fitness máximo en cada generación de ejecuciones típicas para tres algoritmos diferentes (*Square*, *SqNar*, y rPH) al resolver la instancia de 100 vértices del problema MAXCUT ("cut100"). Como se puede ver, la evolución de la mejor solución hasta el momento durante la ejecución es bastante parecida en los tres algoritmos. La diferencia en el comportamiento de los tres algoritmos se puede ver en la parte final de la evolución, donde en los casos de *Square* y *SqNar* la evolución del fitness se atasca durante muchas generaciones antes de encontrar la solución óptima, mientras que el algoritmo adaptativo, debido a la diversidad de la población, encuentra rápidamente la solución óptima.

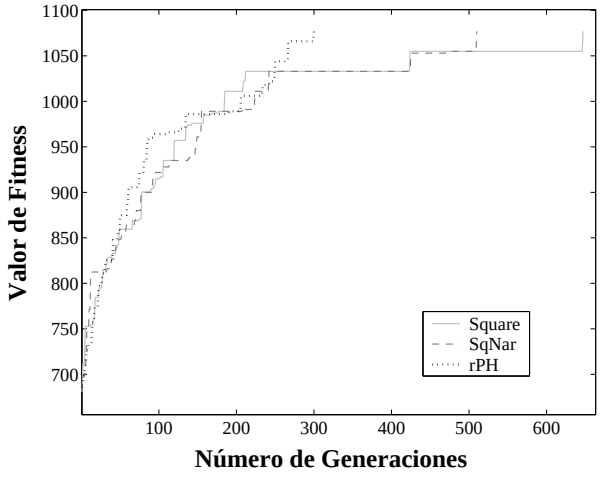


Figura 6.10: Evolución del Mejor valor de fitness para el problema "cut100".

Tabla 6.8: Parametrización utilizada en los cGAs.

	LS	**FRS**	**NRS**	**UC**	**Square**	**Rectangular**	**Narrow**
Forma de la Población	12×12 (Ratio: 0.1832)					8×18 (Ratio: 0.1576)	4×36 (Ratio: 0.0856)
Tamaño de la Población	144 Individuos						
Selección de Padres	Torneo Binario + Torneo Binario						
Recombinación	DPX, $p_c = 1.0$						
Mutación de Bit	Bit-flit, $p_m = 1/n$						
Reemplazo	Reemp_si_no_Peor						
Condición de Parada	Encontrar la solución o realizar 100000 generaciones						

6.5. Modelos Avanzados de cGAs

Presentamos en esta sección una extensión a los trabajos realizados en la Sección 6.4. En concreto, realizaremos un extenso estudio del comportamiento de modelos avanzados de cGAs en los que se combina el uso de actualizaciones asíncronas de la población (ver Capítulo 5) con el uso de rejillas adaptativas. Para comparar los algoritmos propuestos en este estudio se han seleccionado 12 instancias de un único problema, pero de muy elevada complejidad: el problema SAT (descrito en el Apéndice A).

En concreto, además de los tres cGAs síncronos con rejillas estáticas estudiados en la Sección 6.4 (Narrow, Rectangular y Square), hemos estudiado 16 cGAs asíncronos más. Cuatro de ellos utilizan la rejilla estática cuadrada (LS, FRS, NRS, y UC), y los otros son los mismos algoritmos pero utilizando los tres criterios de adaptación presentados previamente en este mismo capítulo: AF (algoritmos AF LS, AF FRS, AF NRS, y AF UC), PH (algoritmos PH LS, PH FRS, PH NRS, y PH UC) y AF+PH (algoritmos AF+PH LS, AF+PH FRS, AF+PH NRS, y AF+PH UC).

Comparamos la eficiencia y eficacia de los algoritmos (los detalles acerca de la parametrización se dan en la Tabla 6.8). Presentamos nuestros resultados en las tablas 6.9 a 6.13 en términos de la tasa de éxito (TE) y el número medio de evaluaciones (NME) necesario para encontrar el óptimo tras 50 ejecuciones independientes. Por tanto, TE es la fracción de ejecuciones en las que se encuentra solución, mientras que NME es el número de evaluaciones realizado en esas ejecuciones exitosas. Es claro por tanto que si TE =0, entonces NME no está definido.

Analizando los valores de TE de todas las tablas, podemos ver que, para todas las instancias de prueba, ninguno de los 19 cGAs estudiado es capaz de encontrar el óptimo en todas las ejecuciones. En particular, para las instancias 6 y 10 no se puede encontrar el óptimo por ninguno de los cGAs propuestos. Pero en cambio la tasa de éxito es en general muy alta en todos los algoritmos, ya que el valor medio de TE para todas las instancias es mayor que el 92.66 % para todos los algoritmos. Además, las diferencias en la TE media de los cGAs para todas las instancias es muy pequeña ya que en todos los casos este valor se encuentra entre 92.66 % de Square y 94.83 % de AF LS y AF NRS. Recordemos que nuestro objetivo no es encontrar la solución óptima a todos los problemas, que se puede hacer por ejemplo incluyendo un paso de búsqueda local en el algoritmo (como se hizo con WSAT [104] y cMA [9] en el Capítulo 9). En cambio, pretendemos en nuestro estudio realizar un análisis comparativo de los algoritmos.

Pasemos a analizar ahora la eficiencia de los cGAs estudiados (medidos en términos de NME). Entre los algoritmos síncronos propuestos, el más eficiente parece ser Square (ver Tabla 6.9), ya que obtiene los valores más bajos de NME en 6 de las 12 instancias probadas. Rectangular es también bastante eficiente, porque es el mejor de los tres algoritmos en 5 de las 12 instancias. En términos de la media de NME para todas las instancias, Square es también el mejor, con una diferencia de sólo el 0.47 % con respecto a Narrow y del 15.67 % con Rectangular. En contraste, Square es el algoritmo con la menor tasa de éxito.

Tabla 6.9: Resultados de los cGAs síncronos con diferentes ratios estáticas.

Inst.		**Narrow**		**Rectangular**		**Square**	
n	#	**TE**	**NME**	**TE**	**NME**	**TE**	**NME**
	1	1.00	8243.2	1.00	7060.5	1.00	7781.8
30	2	1.00	825546.2	1.00	633774.1	1.00	664741.4
	3	1.00	50941.4	1.00	58014.7	1.00	58757.8
	4	1.00	19233.3	1.00	15585.3	1.00	12937.0
40	5	1.00	10449.9	1.00	10245.1	1.00	10765.4
	6	0.76	2149524.2	0.76	2593805.5	0.68	1979966.1
	7	1.00	15209.0	1.00	14863.4	1.00	13970.9
50	8	1.00	64773.1	1.00	58416.6	1.00	66726.7
	9	0.96	873757.3	0.96	2065850.7	0.98	1657989.6
	10	0.54	3850320.6	0.52	3669129.8	0.46	3459593.7
100	11	1.00	162104.3	1.00	155289.6	1.00	128505.6
	12	1.00	334889.0	1.00	300922.9	1.00	264332.2

Tabla 6.10: Resultados de los cGAs asíncronos utilizando la rejilla cuadrada.

Inst.		**LS**		**FRS**		**NRS**		**UC**	
n	#	**TE**	**NME**	**TE**	**NME**	**TE**	**NME**	**TE**	**NME**
	1	1.00	7113.6	1.00	6359.0	1.00	6174.7	1.00	8032.3
30	2	1.00	575925.1	1.00	1259930.9	0.98	1039215.7	1.00	984182.4
	3	1.00	42163.2	1.00	55774.1	1.00	57231.4	1.00	53642.9
	4	1.00	16372.8	1.00	10595.5	1.00	11162.9	1.00	14443.2
40	5	1.00	10794.2	1.00	8657.3	1.00	9383.0	1.00	11295.4
	6	0.72	2202740.0	0.74	2166332.1	0.74	1808885.2	0.70	2280305.8
	7	1.00	12816.0	1.00	13766.4	1.00	13366.1	1.00	15194.9
50	8	1.00	53925.1	1.00	68207.0	1.00	49337.3	1.00	85060.8
	9	0.96	1501134.0	1.00	1194707.5	1.00	1930878.7	1.00	1687541.8
	10	0.64	2753109.0	0.50	3196460.2	0.54	3096293.3	0.56	3341808.0
100	11	1.00	88473.6	1.00	102954.2	1.00	135037.4	1.00	129772.8
	12	1.00	222808.3	1.00	363876.5	1.00	262114.6	1.00	314110.1

Con respecto a los algoritmos asíncronos con rejillas estáticas (mostrados en la Tabla 6.10), podemos ver que LS tiene un coste computacional menor (en términos del valor de NME) que los otros cGAs en 6 de las 12 instancias estudiadas. Tras calcular el valor medio de NME para todas las instancias, vemos que LS obtiene también el mejor (más pequeño) valor, con una diferencia del 11 % con respecto a FRS y NRS, y un 16 % con respecto a UC.

Las tablas 6.11, 6.12, y 6.13 resumen los resultados alcanzados para todos los cGAs asíncronos dinámicos adaptativos estudiados. Si comparamos los algoritmos en términos de la velocidad de convergencia (NME), podemos ver que los algoritmos que utilizan el criterio AF (basados en el fitness medio) tienen los mejores resultados en general, seguidos por los cGAs utilizando AF+PH y PH.

Tabla 6.11: Resultados de los cGAs adaptativos asíncronos basados en el fitness medio de la población.

Inst.		**AF LS**		**AF FRS**		**AF NRS**		**AF UC**	
n	#	**TE**	**NME**	**TE**	**NME**	**TE**	**NME**	**TE**	**NME**
	1	1.00	7807.7	1.00	9432.0	1.00	7021.4	1.00	8208.0
30	2	1.00	1083277.4	1.00	607086.7	0.98	689930.4	0.98	1145969.6
	3	1.00	65085.1	1.00	50561.3	1.00	60215.0	1.00	75939.8
	4	1.00	15431.0	1.00	15215.0	1.00	13219.2	1.00	14912.6
40	5	1.00	10422.7	1.00	9642.2	1.00	9406.1	1.00	10514.9
	6	0.76	2845549.9	0.72	1932720.0	0.78	1899504.0	0.72	1684248.0
	7	1.00	13020.5	1.00	14906.9	1.00	16490.9	1.00	13838.4
50	8	1.00	81121.0	1.00	80824.3	1.00	70272.0	1.00	79856.6
	9	1.00	1235318.4	1.00	1012369.0	1.00	1620679.7	1.00	1318204.8
	10	0.62	2701955.6	0.64	5134549.5	0.62	3653842.1	0.60	4064448.0
100	11	1.00	137065.0	1.00	134231.0	1.00	107089.9	1.00	124655.0
	12	1.00	329385.6	1.00	312030.7	1.00	337331.5	1.00	400495.7

Tabla 6.12: Resultados de los cGAs adaptativos asíncronos basados en la entropía de la población.

Inst.		**PH LS**		**PH FRS**		**PH NRS**		**PH UC**	
n	#	**TE**	**NME**	**TE**	**NME**	**TE**	**NME**	**TE**	**NME**
	1	1.00	6736.3	1.00	8046.7	1.00	8179.2	1.00	8769.6
30	2	1.00	579680.6	0.98	1359127.8	0.98	856085.9	0.98	1012599.2
	3	1.00	65865.6	1.00	70447.7	1.00	55978.6	1.00	55713.6
	4	1.00	13965.1	1.00	11946.2	1.00	16142.4	1.00	17426.9
40	5	1.00	10123.2	1.00	9452.2	1.00	10624.3	1.00	11877.1
	6	0.84	1430502.9	0.76	2368523.4	0.74	1966362.8	0.76	1729242.9
	7	1.00	11678.4	1.00	17487.4	1.00	15370.6	1.00	13475.5
50	8	1.00	64152.0	1.00	79260.5	1.00	58596.5	1.00	52989.1
	9	1.00	1174190.4	0.96	1319100.0	0.98	1439991.2	0.96	1119216.0
	10	0.46	2880651.1	0.60	3695520.0	0.58	4010136.8	0.46	1597855.3
100	11	1.00	109903.7	1.00	151067.5	1.00	109486.1	1.00	177621.1
	12	1.00	345660.5	1.00	328564.8	1.00	363421.4	1.00	346294.1

Para la comparación de las 19 aproximaciones algorítmicas, consideramos apropiado realizar un análisis estadístico para buscar la confidencia estadística en los resultados. Por tanto, aplicamos un análisis de ANOVA o de Kruskal-Wallis en función de si datos siguen una distribución normal o no, respectivamente. Aunque los resultados de todos los algoritmos para cada instancia tienen NME medios similares, hemos observado que los costes computacionales medios no son los mismos en todos los casos. Además, los p-valores nos permiten distinguir dos grupos diferentes, el primero supera ampliamente el valor de confidencia (0.05), y está formado por las instancias: 2, 3, 6, 8, 9, 10 y 12. Por tanto, el otro grupo de instancias, con diferencias estadísticamente significativas, se compone de las instancias: 1, 4, 5, 7 y 11.

Con el fin de resumir los resultados de los 19 cGAs utilizados y obtener conclusiones útiles, presentamos en la Tabla 6.14 dos clasificaciones con los algoritmos ordenados de mejor a peor. Estas clasificaciones se han hecho sumando la posición de los algoritmos en una lista ordenada (de mejor a peor) para cada una de las 12 instancias en términos de NME (cuando TE = 1.0) y TE (en los casos en que TE $\neq$ 1.0), respectivamente.

Tabla 6.13: Resultados de los cGAs adaptativos asíncronos basados en fitness medio y entropía de la población.

Inst.		**AF+PH LS**		**AF+PH FRS**		**AF+PH NRS**		**AF+PH UC**	
n	#	**TE**	**NME**	**TE**	**NME**	**TE**	**NME**	**TE**	**NME**
	1	1.00	7597.4	1.00	7836.5	1.00	7701.1	1.00	8190.7
30	2	0.98	891239.5	1.00	994469.8	1.00	823760.6	1.00	667382.4
	3	1.00	57081.6	1.00	53582.4	1.00	59616.0	1.00	51220.8
	4	1.00	13720.3	1.00	14120.6	1.00	14112.0	1.00	14676.5
40	5	1.00	10195.2	1.00	11111.0	1.00	9878.4	1.00	10661.8
	6	0.68	2503287.5	0.76	2383408.4	0.76	2293200.0	0.68	1847401.4
	7	1.00	13291.2	1.00	13873.0	1.00	13671.4	1.00	15206.4
50	8	1.00	59495.0	1.00	70652.2	1.00	59261.8	1.00	64368.0
	9	1.00	1275027.8	0.98	906215.5	0.98	1229081.1	1.00	1247382.7
	10	0.58	2953301.0	0.58	3537757.2	0.56	2325558.9	0.48	2736672.0
100	11	1.00	154742.4	1.00	79300.8	1.00	131385.6	1.00	167281.9
	12	1.00	458671.7	1.00	299335.7	1.00	342794.9	1.00	298022.4

Por un lado, en la columna TE = 1.0 de la Tabla 6.14 podemos ver que los cGAs asíncronos con ratios estáticas realizan, en general, un menor número de evaluaciones de la función de fitness (mejor eficiencia) que los otros algoritmos estudiados. La excepción es UC, ya que todos los algoritmos que implementan la política de actualización UC se encuentran al fondo de la clasificación. Respecto a los cGAs síncronos, podemos ver que la eficiencia es peor conforme la ratio es menor (poblaciones rectangulares). Debería esperarse que las poblaciones rectangulares retrasen la convergencia del algoritmo, ya que fomentan la exploración.

Tabla 6.14: Clasificación de algoritmos.

TE = 1.0			**TE < 1.0**		
Posición	**Algoritmo**	**Suma de Posiciones**	**Posición**	**Algoritmo**	**Suma de Posiciones**
1	NRS	46	1	AF NRS	6
2	LS	49	1	AF LS	6
3	FRS	67	3	AF+PH FRS	11
4	PH LS	68	4	AF FRS	14
5	Square	79	4	AF+PH NRS	14
6	AF+PH NRS	80	4	PH FRS	14
7	AF NRS	82	7	PH LS	18
7	AF+PH FRS	82	8	LS	19
7	AF+PH LS	82	8	PH NRS	19
10	Rectangular	83	8	AF UC	19
11	AF FRS	91	11	Narrow	20
12	PH NRS	92	12	Rectangular	22
13	PH FRS	95	13	NRS	23
14	AF+PH UC	101	14	FRS	25
15	PH UC	105	15	UC	26
16	AF LS	109	15	PH UC	26
17	Narrow	112	17	AF+PH LS	27
18	UC	114	18	AF+PH UC	33
19	AF UC	116	19	Square	35

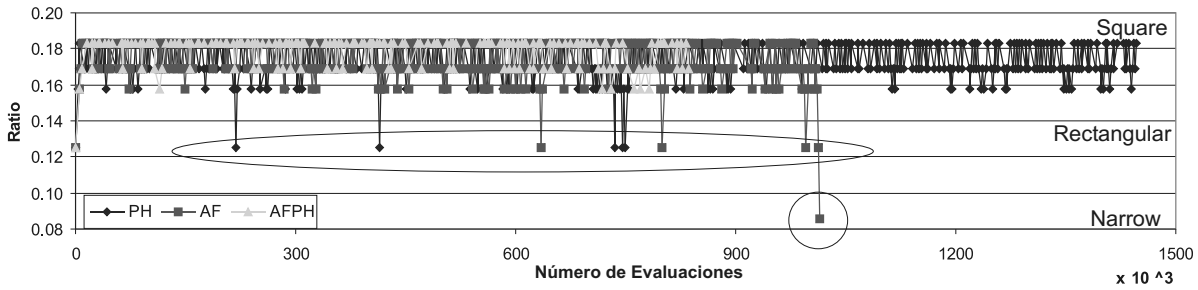


Figura 6.11: Evolución típica del valor de la ratio (instancia 9 de SAT).

Por otro lado, en la columna TE $\neq 1.0$ de la Tabla 6.14 se puede ver cómo los algoritmos adaptativos, y en particular aquellos utilizando el criterio AF, obtienen una tasa de éxito mayor (más efectividad) que los otros algoritmos utilizados. Como en el caso de la clasificación previamente comentada (cuando TE = 1.0), los peores algoritmos son, en general, aquellos que utilizan la política de actualización UC. Finalmente, en contraste con la clasificación hecha para TE = 1.0, los algoritmos síncronos con poblaciones rectangulares están mejor situados que los cuadrados en este caso.

Los algoritmos síncronos estudiados están, en general, en puestos próximos a –o bajo– las posiciones intermedias de la tabla en las dos clasificaciones. Esto significa que los algoritmos asíncronos se comportan, en general, mejor que los síncronos en términos tanto de eficacia (especialmente aquellos usando el criterio AF), como de eficiencia.

Para entender mejor el comportamiento de nuestros cGAs adaptativos, mostramos en la Figura 6.11 la evolución de la fluctuación en el valor de la ratio producido por AF LS, PH LS, y AF+PH LS durante una ejecución típica de una instancia seleccionada del conjunto utilizado.

Todos los algoritmos de la Figura 6.11 muestran una clara tendencia hacia la promoción de la explotación (ratios por encima de 0.18), confirmando las observaciones de la Sección 6.4.5. Esta tendencia general a evolucionar hacia la forma cuadrada es un escenario esperado, porque tras una fase inicial de exploración la búsqueda se enfoca en la explotación de la población. Podemos ver en la Figura 6.11 la existencia de visitas breves y periódicas a ratios menores (bajo 0.13) con el fin de mejorar la exploración del algoritmo, tal y como sucedía en la Sección 6.4 para el caso de los cGAs síncronos adaptativos. Este cambio automático y alternativo entre exploración y explotación en algunos problemas es una característica notable del mecanismo de auto-guía que proponemos, ya que muestra cómo el algoritmo decide automáticamente cuándo explorar y cuándo explotar la población.

6.6. Conclusiones

En este capítulo hemos estudiado el comportamiento de varios GAs celulares sobre un amplio conjunto problemas. La ratio, definida como la relación entre el radio del vecindario y el de la población, representa un único parámetro rico en posibilidades para desarrollar algoritmos eficientes basados en un cGA canónico. Puesto que la población está distribuida sobre una rejilla 2D de forma toroidal, el modelo estudiado está incluido en muchos otros algoritmos similares, así que esperamos que los actuales resultados sean realmente de utilidad a otros investigadores.

Nuestra motivación para este trabajo ha sido avanzar en el conocimiento del equilibrio apropiado entre exploración y explotación para un algoritmo. En concreto, hemos estudiado la influencia de la ratio en el campo de investigación con poblaciones dinámicas estructuradas. Esta característica es, según nuestro conocimiento, utilizada en este estudio por primera vez.

Técnicamente, podemos confirmar los resultados de [28], según los cuales una rejilla estrecha (ratio pequeña) está más indicada para ser utilizada con problemas multimodales y/o engañosos mientras que resulta peor que las rejillas cuadrada y rectangular (ratios elevadas) en el caso de problemas simples y no engañosos. Además, hemos demostrado en este artículo la utilidad del uso de una rejilla intermedia *rectangular* para problemas de optimización combinatoria.

Debemos destacar que cualquiera de los criterios adaptativos desarrollados consiste únicamente en una forma ligera y fácil de obtener técnicas de búsqueda competentes. Una sensación importante que nos deja este trabajo es que se puede guiar la búsqueda fácilmente estructurando la población. Este tipo de modelos son muy eficientes desde un punto de vista numérico, mientras que siguen admitiendo la inclusión de cualquier técnica avanzada desarrollada en el campo de los EAs. Estos algoritmos adaptativos mejoran al resto de los estudiados para todos los problemas propuestos. Las excepciones no tienen en ningún caso relevancia estadística. Nuestras conclusiones son confirmadas utilizando los esquemas de selección por torneo binario y ruleta.

Además, se ha completado este estudio mediante la comparación entre los algoritmos síncronos con ratios estáticas y 16 cGAs asíncronos, cuatro de los cuales utilizan ratios estáticas (en concreto, la rejilla cuadrada), mientras que el resto utilizan ratios adaptativas, siguiendo los mismos criterios que en el caso de los cGAs síncronos adaptativos (AF, PH y AF+PH). De este estudio concluimos que los algoritmos asíncronos con rejillas estáticas cuadradas son los más eficientes de los estudiados, ya que en la clasificación realizada de los algoritmos más eficientes, NRS, LS y FRS son los tres primeros. Por otra parte, los algoritmos asíncronos adaptativos son los más efectivos (especialmente aquellos usando el criterio AF). En general, los cGAs asíncronos (con ratios estáticas y adaptativas) son mejores que los síncronos estáticos para los problemas estudiados.

Capítulo 7

Diseño de Algoritmos Genéticos Jerárquicos Celulares

En este capítulo presentamos un nuevo modelo de cGA en el que se establece una jerarquía entre los individuos que forman la población. De esta manera, los mejores individuos estarán localizados en el centro de la rejilla de la población, mientras que los peores se alojarán en los extremos. Se pretende así acelerar la velocidad de convergencia en los vecindarios de las soluciones más prometedoras de la población (zona céntrica de la malla), mientras que la diversidad tiende a mantenerse en los extremos de la población.

Se examina en este capítulo el efecto de introducir la mencionada jerarquía en la población de un cGA observando la variabilidad de la presión de selección del algoritmo en los diferentes niveles de jerarquía, y comparamos también el rendimiento del nuevo algoritmo jerárquico propuesto con respecto al cGA equivalente sobre un conjunto de problemas de diversas características. De nuestro estudio se desprende que el rendimiento de la nueva aproximación algorítmica es prometedor.

Este capítulo está estructurado de la siguiente forma. En la Sección 7.1 presentamos las principales motivaciones que nos empujan a realizar este estudio. La Sección 7.2 presenta nuestro nuevo algoritmo, HcGA, y además un nuevo operador de selección que hemos diseñado para él. Haremos un análisis minucioso sobre el comportamiento de la presión de selección del algoritmo en la Sección 7.3. La Sección 7.4 contiene los experimentos sobre un conjunto de funciones donde se compara el rendimiento del algoritmo HcGA con un cGA canónico. Finalmente, terminaremos el capítulo mostrando nuestras principales conclusiones.

7.1. Introducción

Mientras que la localización distribuida de la población de un cGA mantiene un alto nivel de diversidad, puede disminuir la velocidad de convergencia del algoritmo debido a que la búsqueda conjunta de la población completa está restringida. Para aumentar la velocidad de convergencia de un cGA, introducimos una jerarquía en la rejilla de la población que ordena a los individuos en función de su fitness actual. Este tipo de población jerárquica ya ha sido aplicado satisfactoriamente a algoritmos de optimización basados en enjambres de partículas (PSO) [155]. Aquí la velocidad de convergencia del algoritmo PSO se aceleró al usar una población jerárquica en la resolución de varios problemas de optimización continuos.

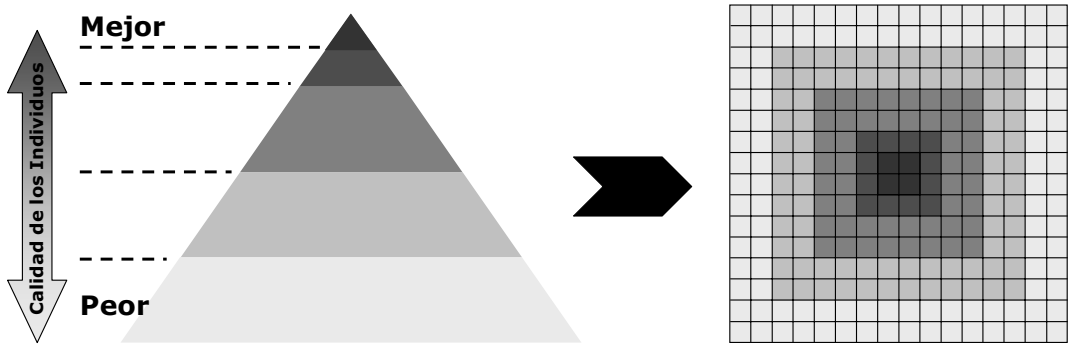


Figura 7.1: Algoritmo genético jerárquico celular – HcGA.

En este trabajo examinaremos el efecto de introducir una jerarquía en un cGA. Hemos llamado a nuestro nuevo algoritmo *Hierarchical Cellular Genetic Algorithm* (HcGA). En HcGA, la jerarquía se establece dentro de la población de un cGA canónico localizando los individuos de acuerdo con su valor de adecuación: cuanto peor sea el fitness del individuo más alejado del centro de la población estará situado. Por tanto, en esta población jerárquica los mejores individuos están localizados en el centro de la rejilla, donde se pueden emparejar con otros individuos de gran calidad. De esta manera se esperan alcanzar mejores soluciones de manera más rápida, manteniendo sin embargo la diversidad proporcionada por una selección de padres restringida localmente, así como la posibilidad de hacer una implementación paralela eficiente. En la Figura 7.1 mostramos cómo esta jerarquía establece una estructura piramidal en la que los peores individuos se sitúan en la base (extremos de la rejilla) y los mejores en el vértice superior (centro de la rejilla).

7.2. El Algoritmo HcGA

En esta sección presentamos el algoritmo cGA jerárquico. En primer lugar esbozaremos brevemente el procedimiento de un GA celular. Después describiremos el orden jerárquico de la población que se introduce en el algoritmo HcGA y cómo obtenemos dicho orden. Finalmente introduciremos un nuevo operador de selección desarrollado específicamente para este algoritmo.

Como ya se ha explicado en detalle con anterioridad en esta tesis, la población de un GA celular se distribuye sobre una rejilla toroidal normalmente de 2 dimensiones (de tamaño $x \times y$), donde un individuo sólo se puede emparejar con individuos de un vecindario localmente restringido. En cada generación, se obtiene un descendiente de cada individuo. En el caso considerado en este capítulo, uno de los padres es el individuo actual y el otro se elige de su vecindario. Cada descendiente se muta con una probabilidad dada, y reemplaza al individuo actual si tiene un mejor valor de fitness. El descendiente (o el individuo actual si es mejor) se almacena en una población temporal auxiliar, que reemplazará a la población actual después de cada generación. En el algoritmo HcGA los individuos de la población son realojados después de cada generación con la operación de intercambio de la jerarquía, que se describe en la sección siguiente.

En el Algoritmo 7.1 se muestra el pseudocódigo de HcGA. Como puede verse, el pseudocódigo presentado es similar al de un cGA canónico (consulte el Capítulo 2 para más detalles del cGA canónico) con la salvedad de la instrucción necesaria para mantener la jerarquía deseada en la población (línea 16), que es ejecutada al final de cada generación.

Algoritmo 7.1 Pseudocódigo de HcGA.

```
1. proc Evoluciona(cga)        // Parámetros del algoritmo en 'cga'
2. GeneraPoblaciónInicial(cga.pobl);
3. Evaluación(cga.pobl);
4. para s ← 1 hasta MAX_PASOS hacer
5.    para i ← 1 hasta cga.ANCHO hacer
6.       para j ← 1 hasta cga.ALTO hacer
7.          vecinos ← CalculaVecindario(cga,posición(i,j));
8.          padres  ← Selección(vecinos);
9.          descendiente ← Recombinación(cga.Pc,padres);
10.         descendiente ← Mutación(cga.Pm,descendiente);
11.         Evaluación(descendiente);
12.         Remplazo(posición(i,j),pobl_auxiliar,descendiente);
13.      fin para
14.   fin para
15.   cga.pobl ← pobl_auxiliar;
16.   OperaciónIntercambio(cga.pobl);
17. fin para
18. fin proc Evoluciona;
```

7.2.1. Jerarquía

La jerarquía se impone en la población del cGA definiendo un centro en la posición $(x/2, y/2)$ y asignando niveles de jerarquía de acuerdo con la distancia al centro. El centro tiene nivel 0 y este nivel se va incrementando conforme aumenta la distancia al centro (véase la Figura 7.2). La jerarquía se actualiza después de cada iteración del cGA y los individuos que tengan un fitness alto se mueven hacia el centro. Este movimiento se lleva a cabo mediante la operación de intercambio, que se explicará a continuación. Observe que la topología de la población sigue siendo toroidal cuando se seleccionan los padres.

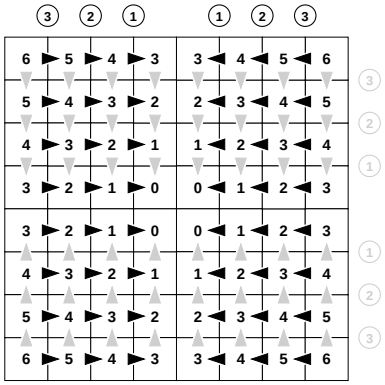


Figura 7.2: HcGA y sus diferentes niveles de jerarquía.

En la Figura 7.2 mostramos cómo se realiza la operación de intercambio. Ésta se aplica entre los individuos que indican las flechas, y en el orden que indican los números que hay fuera de la rejilla. La actualización de la jerarquía la realiza el operador de intercambio alternativamente de manera horizontal (negro) y vertical (gris). Suponemos un número par para las dimensiones x e y de la población, de manera que la población puede ser unívocamente dividida en dos mitades: izquierda (arriba) y derecha (abajo), para el intercambio horizontal (vertical). Observe que esto implica que hay 4 individuos en el centro de la población, que es el nivel más alto de la jerarquía, tal y como se muestra en la Figura 7.2.

A continuación describiremos la operación de intercambio horizontal; la de intercambio vertical se hace de manera análoga. Cada individuo (i, j) de la mitad izquierda se compara con su vecino izquierdo $(i-1, j)$ y si éste último es mejor intercambian sus posiciones. Estas comparaciones se realizan empezando desde el centro de la rejilla hacia el exterior, véase la Figura 7.2. Por tanto, inicialmente se comparan los individuos en las columnas $\frac{x}{2}-1$ y $\frac{x}{2}-2$. Si el valor de fitness de la posición $(i, \frac{x}{2}-2)$ es mejor que el de la posición $(i, \frac{x}{2}-1)$ para $i = 0, \ldots, y$, intercambian las posiciones. Estos pares de comparaciones continúan hasta llegar al borde de la rejilla. De esta manera, un individuo puede avanzar un único nivel, pero en cambio puede perder varios niveles en una única iteración. El caso de la mitad derecha es similar al de la mitad izquierda ya explicado.

7.2.2. Selección por Disimilitud

La jerarquía propuesta fomenta la recombinación de los mejores individuos de la población entre sí. Con respecto a esto, HcGA es similar al GA panmíctico con una selección basada en el fitness. En nuestro algoritmo HcGA esta recombinación selectiva de los mejores individuos de la población ya está incluida en la jerarquía. Por tanto, examinamos un nuevo operador de selección, llamado *selección por disimilitud*, que no está basado en el fitness relativo con respecto a los vecinos, sino que considera la diferencia existente entre las respectivas cadenas que componen los cromosomas de los individuos. Al igual que en la selección por torneo binario (BT), en la selección por disimilitud se seleccionan dos vecinos aleatoriamente, pero a diferencia de BT, en el que se selecciona al mejor individuo, en nuestro caso escogemos aquel cuyo cromosoma difiera más del individuo central. Todos los problemas considerados tienen codificación binaria, por lo que utilizaremos la distancia de Hamming para determinar la disimilitud. En términos generales, el proceso de optimización del algoritmo se garantiza reemplazando un individuo únicamente si el nuevo individuo generado mediante el cruce y la mutación es mejor que el actual.

7.3. Primeros Resultados Teóricos: Tiempo de Takeover

En esta sección, vamos a analizar exhaustivamente las propiedades del algoritmo propuesto realizando un estudio del tiempo de *takeover*, y comparándolo al del cGA canónico. Como se explicó previamente en el Capítulo 5, el tiempo de *takeover* es el tiempo necesario para que un único mejor individuo ocupe completamente la población con copias de sí mismo sólo bajo los efectos de la selección.

Primero veremos un proceso de *takeover* determinista, donde el mejor individuo del vecindario (utilizamos L5) siempre es seleccionado. Más tarde, también tendremos en cuenta la selección BT y la nueva selección de Disimilitud. Inicialmente todos los individuos tienen asignados valores de fitness aleatorios dentro del rango [0,4094] y un único individuo tendrá el máximo valor de fitness 4095. Entonces el cGA que sólo utiliza selección se ejecuta y se va almacenando la proporción de la población que tenga máximo valor de fitness en cada iteración. La rejilla considerada tiene un tamaño de 64×64, por lo que la población consta de 4096 individuos.

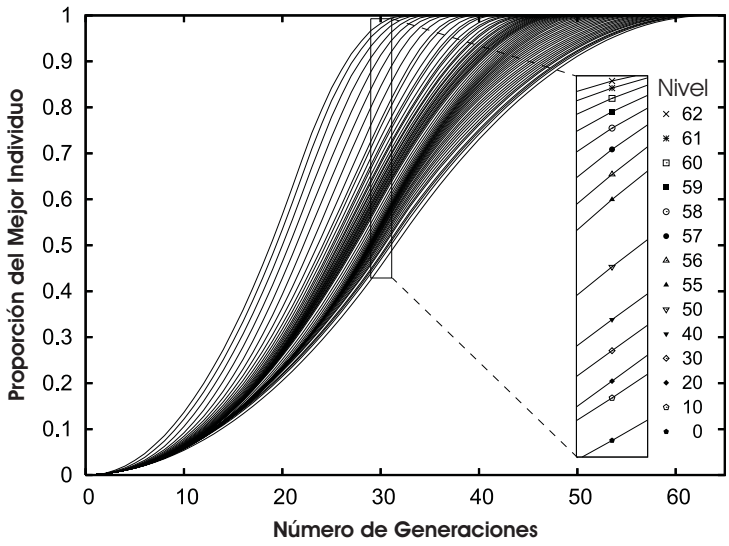


Figura 7.3: Curvas de *takeover* colocando inicialmente el mejor individuo en diferentes niveles de la jerarquía.

Los resultados obtenidos en este estudio tanto con BT como con selección por Disimilitud son calculados como la media de 100 ejecuciones independientes colocando al mejor individuo en una posición inicial aleatoria. Para los experimentos con la selección por Disimilitud, los individuos utilizan una cadena binaria que se corresponde con la representación en 12 bits de su valor de fitness actual.

En HcGA los distintos niveles de jerarquía influyen en el tiempo de *takeover* necesario. Para determinar de manera precisa esta influencia, vamos a utilizar el operador de selección determinista. El mejor individuo se sitúa inicialmente en cada posible posición de la rejilla y se mide el tiempo de *takeover* de estas 4096 posibles disposiciones. Entonces los resultados para todas las posiciones en un nivel específico de la jerarquía se promedian para obtener la tasa de *takeover* en ese nivel en concreto. En la Figura 7.3 se muestran las curvas de *takeover* obtenidas cuando introducimos al individuo en distintos niveles de la jerarquía. La tasa de *takeover* más lenta se obtiene cuando situamos el mejor valor en el centro de la rejilla, en el nivel 0. Esta tasa de *takeover* es idéntica al *takeover* determinista para el cGA canónico. El nivel 62 de la jerarquía consta de los 4 individuos de los extremos de la rejilla, en este caso obtenemos la tasa de *takeover* más rápida. Este aumento de la velocidad del *takeover* con el aumento de la jerarquía es muy normal, tal y como podemos observar en el gráfico detallado de la iteración 30. La razón para esto es que para la selección la topología sigue siendo toroidal, y los individuos adyacentes en los bordes opuestos de la rejilla también adquieren el valor óptimo al inicio de la ejecución. Entonces la operación de intercambio de jerarquía mueve el valor máximo hacia el centro desde distintos lados y por lo tanto la velocidad actual de *takeover* se incrementa.

También medimos el tiempo necesario para completar el *takeover* con las selecciones BT y Disimilitud. El mejor individuo se situó aleatoriamente y los experimentos se repitieron 100 veces. Nuestros resultados se muestran en la Tabla 7.1 (número medio de iteraciones, desviación estándar, mínimo y máximo). Como se puede observar, si colocamos inicialmente el mejor individuo en el centro (HcGA nivel 0) podemos aumentar el tiempo de *takeover* si lo comparamos con un algoritmo cGA canónico. Esto es debido a que una vez que el valor máximo se ha extendido, esta difusión se retrasará hasta que todos los individuos

Tabla 7.1: Tiempos de *takeover* para algoritmos con selecciones BT y Disimilitud.

Algoritmo	**Media**	**Desv. Est.**	**Min.** – **Max.**
cGA BT	75.2	± 1.5	72.0– 80.0
cGA Dis	78.7	± 1.7	75.0– 83.0
HcGA BT	71.0	± 6.3	48.0– 81.0
HcGA Dis	79.6	± 3.8	71.0– 89.0
HcGA nivel 0	81.5	± 1.5	78.0– 87.0
HcGA nivel 62	46.3	± 2.1	42.0– 57.0

centrales tengan el mejor valor de fitness, si no es así, cambiará hacia el centro de nuevo. En ambos algoritmos, la utilización de la selección BT aumenta la presión de selección con respecto a la selección por Disimilitud. Tras aplicar el test de Kruskal-Wallis (en algunos casos los datos no tenían distribución normal) a los resultados en la Tabla 7.1, pudimos comprobar que existen diferencias estadísticamente significativas con un nivel de confianza del 95 %.

7.3.1. Ajuste de las Curvas de Takeover

El tiempo de *takeover* disminuye cuando el nivel del mejor individuo introducido aumenta, tal y como se muestra en la Figura 7.3. Para poder observar de manera más precisa esta distinción, ajustamos las curvas de *takeover* con una función paramétrica, de forma que podamos comparar simplemente los respectivos parámetros de ajuste.

En este estudio preliminar utilizamos dos de las formulaciones más sencillas para modelar el crecimiento de los HcGAs. Son la ecuación logística LOG y una formulación sencilla cuadrática; ambas están recogidas en el Capítulo 5. Las dos curvas de crecimiento se controlan mediante un parámetro a, y se repiten a continuación:

$$LOG(t) = \frac{1}{1 + (\frac{1}{N(0)} - 1)\, e^{-at}} \ . \tag{7.1}$$

$$N(t) = \begin{cases} a\, t^2 & \text{si } t < T/2 \ , \\ -a\, (t - T)^2 + 1 & \text{en otro caso} \ . \end{cases} \tag{7.2}$$

Ajustamos el parámetro a a las curvas de *takeover* obtenidas para los diferentes niveles de la jerarquía. Esto se hace minimizando el error cuadrático medio (MSE) entre los puntos de datos observados y la curva parametrizada. En la Figura 7.4 se presentan las curvas de *takeover* deterministas y las curvas de ajuste al introducir el mejor individuo en el nivel 0 y en el nivel 62.

Al igual que las curvas de *takeover* para diferentes niveles, los parámetros para ajustar dichas curvas también están bien ordenados. En la Figura 7.5 mostramos los valores del parámetro a que devuelve el ajuste de los distintos niveles con la Ecuación 7.3.1. Esto se ha hecho para el *takeover* determinista y para el *takeover* con selección BT. Los niveles de crecimiento para el *takeover* determinista están completamente ordenados y, con alguna excepción, también para el *takeover* del HcGA utilizando BT a aumenta conforme aumenta el nivel de la jerarquía.

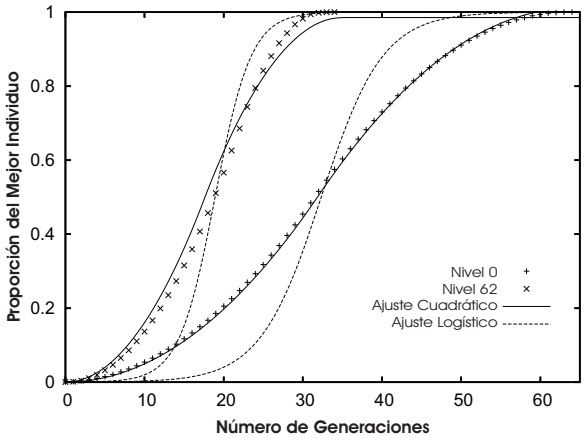


Figura 7.4: Ajuste de curvas de *takeover*.

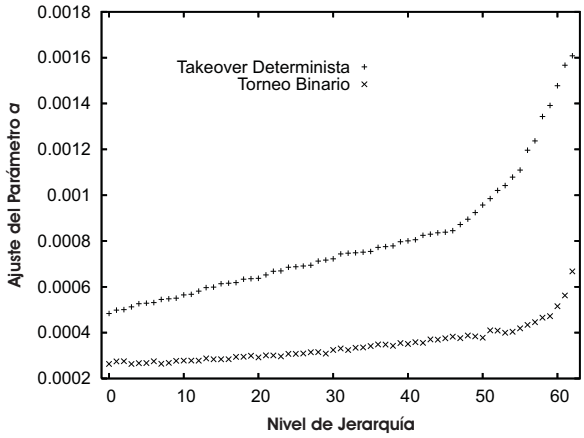


Figura 7.5: El parámetro a para diferentes niveles.

7.4. Experimentos Computacionales

En esta sección presentamos los resultados de nuestras pruebas en un conjunto determinado de problemas seleccionados con distintas características: engañosos, multimodales, con restricciones o generadores de problemas. En concreto, los problemas propuestos son Onemax (con 500 variables), MMDP, P-PEAKS (100 óptimos) y MTTP (instancias de 20, 100 y 200 variables). Todos estos problemas están descritos en el Apéndice A. A continuación, procedemos a comentar nuestros resultados.

Nuestro objetivo es comparar el comportamiento de nuestra nueva propuesta, HcGA, frente al cGA. Utilizamos la misma parametrización para ambos algoritmos (descrita en la Tabla 7.2), y se han probado tanto la selección BT como Disimilitud. La población está formada por 400 individuos dispuestos en una rejilla de 20×20, y el vecindario utilizado es L5. En todos nuestros experimentos, uno de los padres es el propio individuo y el otro padre se escoge mediante las selecciones BT o Disimilitud (asegurando que los dos padres sean distintos). Un individuo se sustituye si el fitness generado del nuevo individuo es igual o mejor. El método de recombinación usado es el cruce de dos puntos (DPX), y el descendiente escogido es aquel que tiene la mayor parte del mejor padre. Las probabilidades de mutación y cruce son 1.0, mientras que la mutación de un gen se da con probabilidad $1/\#bits$, donde $\#bits$ es la longitud del cromosoma. Para conseguir una alta confianza estadística, todos los resultados presentados son promedios de 100 ejecuciones independientes, y se ha aplicado el test estadístico de análisis de varianza, ANOVA (o Kruskal-Wallis si los datos no tienen una distribución normal) sobre el número de evaluaciones realizado por todos los algoritmos en cada problema. (Se considera un nivel de confianza del 95 %.)

Tabla 7.2: Parametrización usada en nuestros algoritmos.

Tamaño de la Población	400 Individuos (20×20)
Vecindario	L5
Selección de los Padres	Individuo Actual + (BT o Disimilitud)
Recombinación	DPX, $p_c = 1.0$
Mutación de Bit	Bit-flip, $p_m = 1/\#bits$
Reemplazo	Reemp_si_no_Peor
Criterio de Parada	Encontrar el Óptimo o Alcanzar 2500 Generaciones

Tabla 7.3: Resultados de las diferentes funciones de prueba para dos HcGAs y los cGAs equivalentes.

Problema	**Algoritmo**	**Éxito**	**Media**	**Desv. Est.**	**Min.–Max.**	**Test**
Onemax	cGA BT	100 %	129.4	± 7.3	111.2– 145.2	+
	cGA Dis	100 %	140.7	± 8.1	121.6– 161.2	
	HcGA BT	100 %	94.1	± 5.0	83.2– 106.4	
	HcGA Dis	100 %	103.1	± 5.6	90.4– 116.8	
MMDP	cGA BT	67 %	202.4	± 154.7	120.8– 859.2	+
	cGA Dis	97 %	179.8	± 106.3	116.8– 846.0	
	HcGA BT	55 %	102.6	± 76.1	68.8– 652.8	
	HcGA Dis	92 %	122.3	± 111.7	73.2– 837.6	
P-Peaks	cGA BT	100 %	41.9	± 3.0	32.0– 48.4	+
	cGA Dis	100 %	52.9	± 5.2	38.4– 66.0	
	HcGA BT	100 %	47.2	± 8.6	30.8– 71.2	
	HcGA Dis	100 %	81.1	± 17.1	45.2– 130.8	
MTTP-20	cGA BT	100 %	5.1	± 1.2	1.6– 8.0	+
	cGA Dis	100 %	6.0	± 1.3	2.0– 9.2	
	HcGA BT	100 %	4.7	± 1.1	1.6– 7.2	
	HcGA Dis	100 %	5.5	± 1.2	2.8– 8.0	
MTTP-100	cGA BT	100 %	162.2	± 29.3	101.6– 241.6	+
	cGA Dis	100 %	174.6	± 26.3	96.4– 238.8	
	HcGA BT	100 %	138.3	± 35.4	62.0– 245.6	
	HcGA Dis	100 %	132.4	± 26.2	64.0– 186.8	
MTTP-200	cGA BT	100 %	483.1	± 55.3	341.6– 632.4	+
	cGA Dis	100 %	481.0	± 71.6	258.8– 634.8	
	HcGA BT	100 %	436.2	± 79.7	270.4– 631.2	
	HcGA Dis	100 %	395.3	± 72.6	257.6– 578.8	

En la Tabla 7.3 se presentan los resultados que hemos obtenido en los problemas de prueba. Específicamente, mostramos la tasa de éxito (número de ejecuciones en las que se encuentra el óptimo), y algunas medidas del número de evaluaciones realizadas para encontrar el óptimo, como el valor medio, la desviación estándar, y los valores máximo y mínimo, donde el símbolo '+' significa que existen diferencias estadísticamente significativas. Los algoritmos evaluados son HcGA y el cGA equivalente, ambos con selecciones BT y Disimilitud.

Como se puede ver en la Tabla 7.3, tanto el cGA como el HcGA encuentran el valor óptimo en cada ejecución para todos los problemas, con la excepción de MMDP. Para este problema, el cGA muestra una tasa de éxito levemente mejor que el HcGA. Con respecto al número medio de evaluaciones necesarias para alcanzar el óptimo, el algoritmo jerárquico siempre mejora al cGA, excepto para el problema P-PEAKS. Por consiguiente, el uso de una población jerarquizada nos permite acelerar la velocidad de convergencia del algoritmo hacia el óptimo, mientras que mantiene la interesante gestión de la diversidad del cGA canónico.

Si comparamos ahora los resultados de los algoritmos cuando utilizan los dos esquemas de selección estudiados, notamos que con la selección de Disimilitud la tasa de éxito para el problema MMDP se puede incrementar tanto en cGA como en HcGA.

En las figuras 7.6 a 7.11 mostramos la probabilidad que tienen los algoritmos de encontrar el óptimo en cada generación para todos los problemas estudiados. Esta probabilidad la hemos calculado como el valor medio de las 100 ejecuciones de la mejor solución conocida en cada generación, normalizada con el valor de la solución óptima. De esta manera, estamos comparando la presión de selección de los distintos algoritmos para todos los problemas.

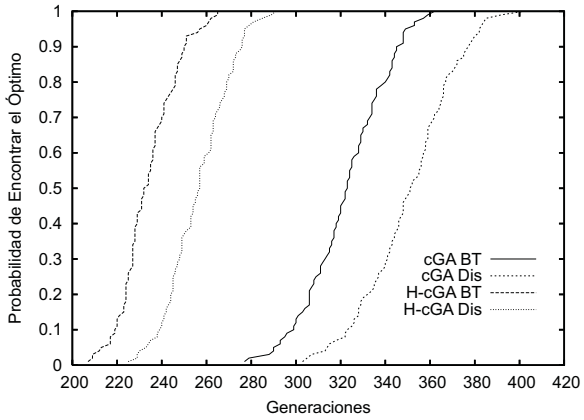


Figura 7.6: Onemax – Probabilidad de encontrar el valor óptimo en cada iteración.

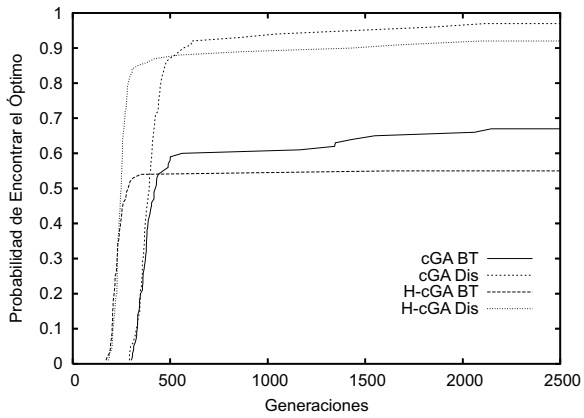


Figura 7.7: MMDP – Probabilidad de encontrar el valor óptimo en cada iteración.

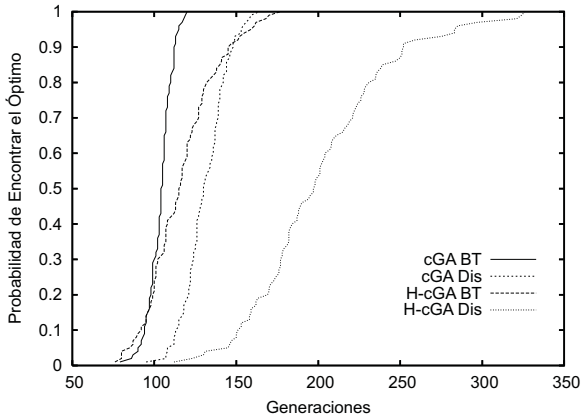


Figura 7.8: P-Peaks – Probabilidad de encontrar el valor óptimo en cada iteración.

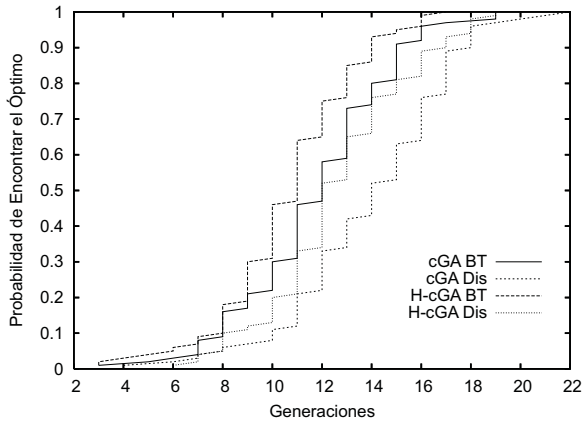


Figura 7.9: "mttp20" – Probabilidad de encontrar el valor óptimo en cada iteración.

Las gráficas mostradas en las figuras 7.6 a 7.11 confirman las conclusiones que se han obtenido de la Tabla 7.3. Como podemos ver en la Figura 7.7, la probabilidad de alcanzar el óptimo en menos de 2500 generaciones para MMDP no es en ningún caso del 100 %, pero efectivamente es más alta en el caso de los cGAs que en el de los HcGAs. En el resto de los problemas la probabilidad de alcanzar el óptimo es siempre del 100 %. En el caso del problema P-PEAKS (Figura 7.8), también presentan una convergencia más rápida los cGAs frente a los HcGAs, pero en el resto de los problemas estudiados, los HcGAs convergen más rápidamente que los cGAs al óptimo.

En la Figura 7.12 dibujamos el número de evaluaciones esperado, definido como el número medio de evaluaciones dividido por la tasa de éxito. Los resultados mostrados han sido divididos por el número esperado de evaluaciones para el cGA con selección por torneo binario.

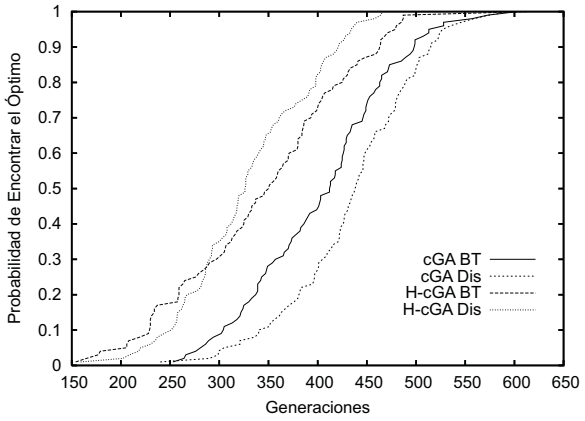


Figura 7.10: "mttp100" – Probabilidad de encontrar el valor óptimo en cada iteración.

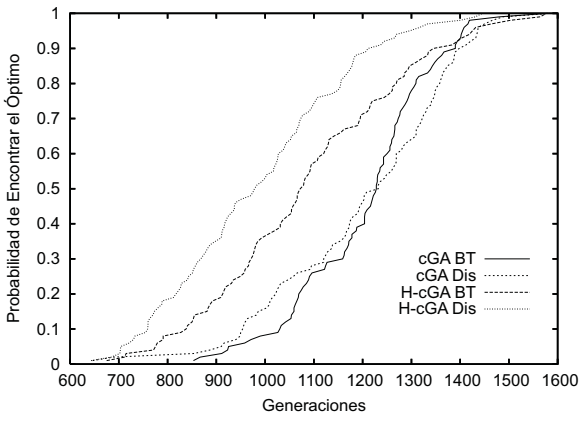


Figura 7.11: "mttp200" – Probabilidad de encontrar el valor óptimo en cada iteración.

El número esperado de evaluaciones se incrementa cuando se utiliza la selección por Disimilitud comparado al algoritmo equivalente con BT en 7 de los 12 problemas estudiados. Para el cGA la selección por Disimilitud puede reducir el número de evaluaciones necesarias únicamente para el problema MMDP. Pero se prueba que en el caso del HcGA puede ser útil también para las dos grandes instancias de MTTP. En general, el número esperado de evaluaciones es menor para las dos versiones estudiadas de HcGA.

Finalmente, para ilustrar los efectos de usar una población jerárquica en el cGA, mostramos una ejecución de ejemplo del cGA (arriba) y del HcGA (abajo) en la Figura 7.13. Las figuras son instantáneas de la población tomadas cada 50 iteraciones para el problema MMDP hasta que se encuentra el óptimo (iteración 383 para cGA y 233 para HcGA). Cuanto más oscuro esté coloreado un individuo, mejor (mayor) es su valor de fitness; el individuo blanco en la última imagen representa la solución óptima al problema. Como se puede observar, el HcGA se centra rápidamente en soluciones prometedoras, mientras que a la vez se mantienen las distintas soluciones de mala calidad en los bordes de la malla jerárquica.

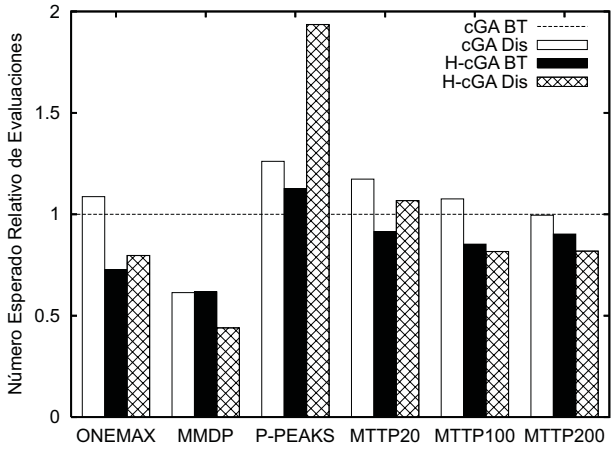


Figura 7.12: Número esperado de evaluaciones necesarias para alcanzar el óptimo, mostrado en relación a los pasos necesarios para el cGA, en los diferentes problemas de estudio.

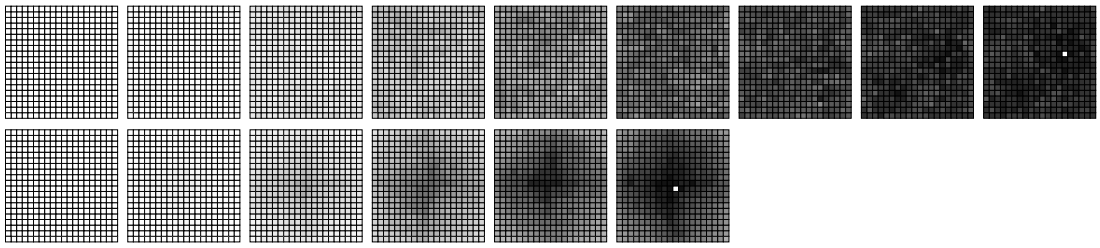


Figura 7.13: Evolución de la población para un cGA (arriba) y para un HcGA (abajo).

7.5. Conclusiones

En este capítulo hemos presentado un nuevo algoritmo llamado HcGA o cGA jerárquico. Básicamente, el HcGA se basa en la idea de establecer una jerarquía entre los individuos de una población de un algoritmo cGA canónico. En este modelo jerárquico localizamos simultáneamente distintos niveles de la relación exploración/explotación del algoritmo en zonas distintas de la población. Hemos estudiado la influencia de los distintos niveles de la jerarquía en el comportamiento del algoritmo examinando los respectivos tiempos de *takeover* (que son indicativos de la presión de selección de los algoritmos).

Hemos comparado el algoritmo HcGA, utilizando dos métodos de selección diferentes, con los cGAs equivalentes y el algoritmo jerárquico funciona mejor en casi todas las funciones de prueba. La nueva selección propuesta de Disimilitud no es útil en todos los escenarios, pero en el caso del algoritmo HcGA mejora el rendimiento para las funciones MMDP y MMTP. Esta selección ofrece mayor diversidad en la población pero, como consecuencia, normalmente la convergencia es más lenta.

Capítulo 8

Diseño de Algoritmos de Estimación de Distribución Celulares

Los Algoritmos de Estimación de Distribución (EDAs) [177, 204] son una familia alternativa a los tradicionales EAs en los cuales se utiliza otro tipo de operador de variación. Las sucesivas generaciones de individuos se crean utilizando estimaciones de distribuciones que se observan en la población actual en lugar de hacer evolucionar la población con los operadores de variación típicos (como la recombinación o la mutación) usados en otros EAs. Por tanto, la característica principal que distingue a los EDAs de los EAs clásicos es que los EDAs aprenden de la interacción entre variables básicas (consideradas pilares fundamentales) en los problemas a resolver. Al mismo tiempo, éste es el principal inconveniente de los EDAs debido a la complejidad que supone realizar esta tarea de aprendizaje y simulación.

La aplicación de los EDAs a los problemas de optimización ha sido intensa durante la última década [177]. La motivación es, en parte, que en muchos de los resultados mostrados los EDAs mejoran a otros EAs como los algoritmos genéticos [52, 177, 230]. Debido al alto coste computacional de muchos EDAs, el actual estado del arte del campo necesita el desarrollo de estrategias nuevas y más potentes para implementarlos.

Como ya se ha comentado varias veces a lo largo de esta tesis, descentralizar la población proporciona en muchos casos un mejor muestreo del espacio de búsqueda, lo cual aumenta el comportamiento numérico del algoritmo. En este capítulo, proponemos una descentralización de la población de los EDAs dividiéndola en muchas pequeñas sub-poblaciones colaboradoras, de manera que se les permita sólo interactuar con las sub-poblaciones vecinas para calcular la estimación de distribución. Esta estimación se utiliza para obtener la siguiente generación de la sub-población que estamos considerando. Los resultados son los llamados EDAs celulares (cEDAs) [193, 223]. Observe que MÍDELA [48] puede parecer como una aproximación parecida a nuestra propuesta, pero en MÍDELA el espacio de soluciones está dividido en grupos para obtener la estimación de distribución de cada grupo, y entonces todas las estimaciones obtenidas se mezclan para obtener una estimación única que calcule la siguiente población completa. En cambio, nosotros utilizamos una estimación de distribución para cada población en nuestra aproximación.

La principal contribución de este capítulo es un estudio comparativo sobre el comportamiento de nuestra nueva propuesta, los cEDAs, y sus EDAs equivalentes. Para este estudio, afrontamos un conjunto grande de problemas discretos con MUDA (un EDA simple) y 4 versiones celulares distintas de MUDA (cUMDAs). Este estudio va más allá del que se había elaborado en [193], que comparaba un

MUDA y un cUMDA en un único problema sencillo y que confirma los resultados que se obtienen en este trabajo. Como una segunda contribución, extendemos el estudio de la presión de selección de un algoritmo al campo de los EDAs para incluir una breve comparación con el comportamiento teórico de los algoritmos propuestos. Según nuestro conocimiento, esta es la primera vez que este estudio de la presión de la selección se aplica a un EDA, y constituye una buena aproximación empírica para estudiar el comportamiento de un algoritmo.

El capítulo se estructura de la siguiente manera. En la próxima sección se presenta una breve introducción al campo de los EDAs. En la Sección 8.2 describimos la aproximación del EDA celular estudiado en este capítulo. Tras esto, presentamos nuestros resultados en la Sección 8.3, así como una explicación de los algoritmos utilizados. Finalmente, terminaremos el capítulo dando nuestras conclusiones.

8.1. Algoritmos de Estimación de Distribución

En esta sección presentamos una breve descripción de un EDA genérico, tras lo que resumimos los principales tipos de EDAs que se pueden encontrar en la literatura. En el Algoritmo 8.1 presentamos un pseudocódigo de un EDA. Como se puede ver, comienza generando aleatoriamente la población inicial (línea 2), que irá evolucionando iterativamente hasta que se cumpla la condición de terminación. Esta condición de terminación consiste generalmente en encontrar la solución óptima al problema o realizar un número máximo de evaluaciones de función. La forma en que la población evoluciona es, precisamente, la principal diferencia entre la familia de EDAs y otros tipos de EAs tradicionales (líneas 5 y 6). La nueva generación de individuos se calcula como sigue. Se seleccionan de entre todos los individuos de la población únicamente M individuos (con $M \leq N$, siendo N el tamaño de la población) –línea 4. Entonces, el EDA extrae información estadística global de este conjunto de M soluciones padre para construir un modelo de distribución de probabilidad de soluciones prometedoras $p^s(x,t)$, basada en la información extraída (línea 5). Tras realizar dicha estimación de probabilidad, se generan N nuevas soluciones del modelo construido, que reemplazan a las soluciones de la población actual (en la generación t) completa o parcialmente para constituir la nueva población (en la generación $t+1$) –línea 6. Los pasos más costosos de los EDAs son, precisamente aquellos que se realizan en estos dos últimos pasos (líneas 5 y 6): la estimación de la distribución $p^s(x,t)$, y la generación de nuevos puntos siguiendo esta distribución.

Generalmente, los EDAs trabajan con cadenas de variables discretas o continuas de longitud fija. Cuando las variables son discretas, la probabilidad de distribución se representa mediante redes Bayesianas de diversas complejidades, donde los nodos son las variables y los ejes representan las relaciones entre las variables. Otra forma de representación de la distribución de probabilidades en el caso discreto son las redes de Markov [245]. Por otro lado, en el caso de utilizar variables pertenecientes al espacio continuo se suelen utilizar comúnmente redes Gaussianas para estimar la distribución.

Algoritmo 8.1 Pseudocódigo de un EDA típico.

1. $t \leftarrow 1$;
2. Generar $N >> 0$ puntos aleatoriamente;
3. **mientras** no se cumpla condición de terminación **hacer**
4. Seleccionar $M \leq N$ puntos según un método de selección;
5. Estimar la distribución $p^s(x,t)$ del conjunto seleccionado;
6. Generar N puntos nuevos según la distribución estimada $p^s(x,t)$;
7. $t \leftarrow t+1$;
8. **fin mientras**

Podemos establecer una clasificación de EDAs en función del tipo de interacción entre alelos (variables) permitido en el modelo de la distribución de probabilidad [177]. Por tanto, distinguimos entre EDAs sin dependencias entre variables, con interacciones entre pares de variables, o con interacciones múltiples:

1. **Sin dependencias**. Ésta es la forma más simple de estimar la distribución de soluciones prometedoras. En este tipo de EDA se asume que todas las variables son independientes entre sí. Matemáticamente, la estimación de distribución para EDAs sin dependencias entre variables es $p^s(x,t) = \prod_{i=1}^{n}(x_i,t)$. Estos algoritmos sin dependencias son apropiados para resolver problemas lineales, en los que las variables no interaccionan entre sí, pero fallan para los problemas con dependencias fuertes entre variables, ya que no las tienen en cuenta. Algunos ejemplos existentes de este tipo de EDAs son los *Univariate Marginal Distribution Algorithm*, UMDA [202, 207], o el algoritmo *Population Based Incremental Learning*, PBIL [34].

2. **Dependencias entre pares de variables**. Este tipo de EDA fue el primero en asumir que las variables de un problema no son independientes. Estos algoritmos calculan la estimación de distribución teniendo en cuenta las dependencias existentes entre pares de variables. Estos EDAs reproducen una mezcla de conjunto de variables básicas de orden dos de forma muy eficiente; por tanto, son especialmente adecuados para resolver problemas lineales y cuadráticos. De este tipo de algoritmos, destacaremos el *Mutual Information Maximization Algorithm for Input Clustering*, MIMIC [73], MIMIC_{C}, una adaptación de MIMIC para el dominio continuo [175, 174], y el *Bivariate Marginal Distribution Algorithm*, BMDA [230].

3. **Dependencias múltiples**. Considerar dependencias entre pares de variables es usualmente insuficiente para resolver problemas con dependencias multivariables o altamente epistáticos, por lo que la investigación en este área continuó con problemas más complejos. En el caso de las múltiples dependencias se consideran todas las posibles dependencias entre las variables, sin restricciones en la complejidad requerida por el algoritmo de aprendizaje. Estos algoritmos alcanzan un buen rendimiento para un rango amplio de problemas descomponibles, aunque pueden encontrar dificultades cuando los problemas son descomponibles en términos de órdenes de frontera. Ejemplos de esta clase de EDA son el *algoritmo de optimización Bayesiano* (*Bayesian Optimization Algorithm*, BOA) [229], el *algoritmo de distribución de aproximación con poli-árbol* (*Polytree Approximation of Distribution Algorithm*, PADA) [257], y *algoritmo de estimación de redes Gaussianas* (*Estimation of Gaussian Networks Algorithm*, EGNA_{BGe}) [174, 175].

En la Figura 8.1 mostramos gráficamente las relaciones entre variables que se tienen en cuenta por los diferentes EDAs para modelar la probabilidad de distribución sin dependencias (MUDA), con dependencias entre pares de variables (MIMIC), y con múltiples dependencias (BOA). Además, existen otras estructuras diferentes, como el caso del ECGA [176], en el que las variables se dividen en un conjunto de grupos mutuamente independientes, siendo cada uno de estos grupos tratados como un todo.

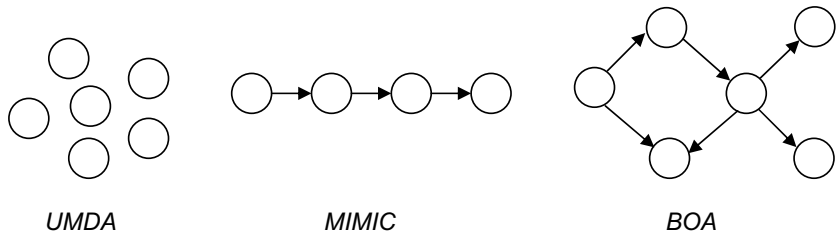


Figura 8.1: Relaciones entre variables en algunos EDAs.

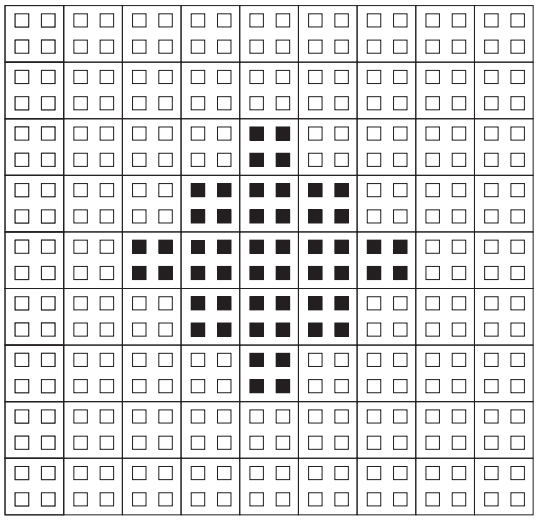


Figura 8.2: Un cEDA con un vecindario C13 y la forma de la población $2 \times 2 - 9 \times 9$.

8.2. Algoritmos de Estimación de Distribución Celulares

Los EDAs celulares se introdujeron en [223] como una versión descentralizada de los EDAs, y también como una generalización de los modelos celulares desarrollados para otros algoritmos evolutivos [33, 195]. La población de un cEDA se descentraliza dividiéndola en muchas pequeñas sub-poblaciones que colaboran entre sí (llamadas *celdas* o *algoritmos miembros*), localizadas en una rejilla toroidal, e interactuando sólo con las sub-poblaciones vecinas. Una característica distintiva de este tipo de algoritmos es que la selección está descentralizada en el nivel de los algoritmos miembros, mientras que normalmente en las selecciones de otros EAs celulares suele ocurrir en el nivel de recombinación.

La organización de los cEDAs se basa en la tradicional estructura en dos dimensiones de vecindarios solapados (véase el Capítulo 2 para más detalles). Esta estructura se entiende mejor como dos mallas: una que consistiría en cadenas y la otra consistiría en conjuntos disjuntos de cadenas (celdas). La Figura 8.2 muestra una población global de 18×18 cadenas (cuadrados pequeños) divididos en una rejilla toroidal de 9×9 celdas (cuadrados grandes) que contienen 4 cadenas cada uno. El vecindario utilizado es el llamado C13, compuesto por la sub-población que estamos considerando más sus 12 celdas más cercanas (medido con la distancia de Manhattan). Adoptaremos la misma notación usada en [223] para describir la forma de la población, consistente en la forma de las celdas en términos de cadenas, más la forma de toda la población teniendo en cuenta las celdas (compuestas por una o más cadenas). Por ejemplo, siguiendo con esta nomenclatura, la rejilla de la Figura 8.2 está etiquetada como $2 \times 2 - 9 \times 9$.

En el Algoritmo 8.2 presentamos el pseudocódigo de nuestra aproximación a un EDA celular. Cada iteración de un cEDA consiste exactamente en una iteración de todos los algoritmos miembros. Cada uno de estos algoritmos miembros es responsable de actualizar exactamente una sub-población, y se hace aplicando un modelo local de EDA a la población formada por sus cadenas y aquellas de sus sub-poblaciones vecinas (líneas 5 a 7). En la implementación del cEDA llevada a cabo aquí las sucesivas poblaciones se reemplazan de una vez (línea 10), de manera que los nuevos individuos generados por el aprendizaje local y los pasos de muestreo se sitúan en una población temporal (línea 8). En este trabajo

Algoritmo 8.2 Pseudocódigo de un cEDA simple.

1. $t \leftarrow 1$;
2. Generar $N >> 0$ puntos aleatoriamente;
3. **mientras** no se cumpla condición de terminación **hacer**
4. **para** cada celda **hacer**
5. Seleccionar localmente $M \leq$ *TamañoDe*(Vecindario) $\times$ *TamañoDe*(Celda) cadenas del vecindario según algún criterio de selección;
6. Estimar la distribución $p^s(x,t)$ de estas M cadenas seleccionadas;
7. Generar *TamañoDe*(Celda) nuevos puntos según la distribución $p^s(x,t)$;
8. Insertar los puntos generados en la misma celda de una población auxiliar;
9. **fin para**
10. Reemplazar la población actual por la auxiliar;
11. Calcular y actualizar estadísticas;
12. $t \leftarrow t + 1$;
13. **fin mientras**

hemos propuesto el uso de una política de actualización de celdas síncrona, aunque también podríamos haber utilizado alguna política asíncrona (consulte el Capítulo 2). El aspecto asíncrono no lo vamos a estudiar aquí por las muchas implicaciones y las numerosas políticas asíncronas existentes.

En la etapa de reemplazo (línea 10), la población antigua puede ser tenida en cuenta (reemplazando una cadena si la nueva es mejor) o no (siempre se añade la nueva cadena a la siguiente población). El primer caso (llamado *elitismo*) es el preferido para este estudio. Finalmente, el cálculo de las estadísticas básicas (línea 11) raramente se encuentra en los pseudocódigos de otros autores en el campo de los EAs. De todas formas, se puede utilizar para monitorizar el algoritmo y decidir los cambios en la búsqueda adaptativa cuando sea necesario (como se propone en el Capítulo 6).

Un caso crítico en un cEDA es el cálculo del modelo de probabilidad debido al alto coste computacional que normalmente supone. Referimos al lector a [193, 223] donde se explican distintos esquemas de aprendizaje alternativo para los EDAs celulares.

8.3. Experimentación

Presentamos en esta sección los experimentos que hemos realizado en este trabajo. Para llevar a cabo nuestras pruebas, se ha seleccionado un conjunto de problemas compuesto por cinco problemas que tienen distintas características: OneMax (de 1000 variables), Plateau, IsoPeak, P-PEAKS (con 100 óptimos) y el MTTP (instancia "mttp100"). En el Apéndice A presentamos en detalle cada uno de estos problemas.

En la Sección 8.3.1 explicamos brevemente el funcionamiento del algoritmo MUDA, el EDA simple que hemos escogido para nuestro estudio. Después, en la Sección 8.3.2 mostraremos y analizaremos los resultados de las pruebas realizadas.

8.3.1. Algoritmo de Distribución Marginal de una sola Variable

El Algoritmo de Distribución de una sola Variable (MUDA) se presentó por primera vez por Müehlenbein y Paaß en [204]. MUDA es uno de los algoritmos más simples en la familia de los EDAs. Un pseudocódigo de MUDA se presenta en el Algoritmo 8.3. En MUDA, se considera que las variables son

Algoritmo 8.3 Algoritmo de distribución marginal de una sola variable – MUDA.

1. $t \leftarrow 1$;
2. Generar $N >> 0$ puntos aleatoriamente;
3. **mientras** no se cumpla condición de terminación **hacer**
4. Seleccionar $M \leq N$ puntos según un método de selección;
5. Calcular las frecuencias marginales $p^s(x_i, t)$ de un conjunto seleccionado;
6. Generar N puntos nuevos según la distribución $p^s(x,t) = \prod_{i=1}^{n} p^s(x_i, t)$;
7. $t \leftarrow t + 1$;
8. **fin mientras**

independientes unas de otras, por tanto, no existen dependencias entre ellas. Esto implica que no necesitamos aprender la estructura del modelo (es conocida), sino únicamente la frecuencia marginal de cada variable. La generación actual evoluciona hacia la nueva calculando los valores de las frecuencias de las variables de cada posición en el conjunto de soluciones prometedoras (líneas 4 y 5). Estas frecuencias se utilizan para calcular las nuevas soluciones (línea 6), que reemplazarán a las antiguas. Debido a su simplicidad, el algoritmo es muy eficiente (converge rápidamente) y su comportamiento es particularmente bueno para los problemas lineales.

Hemos utilizado parametrizaciones comunes en nuestras pruebas para poder hacer comparaciones significativas entre los algoritmos estudiados. Mostramos los detalles en la Tabla 8.1. Como se puede ver, hemos definido distintas configuraciones de cUMDAs. Estos cUMDAs difieren en la población y el vecindario utilizados. La población siempre se compone de 400 individuos, pero en el caso de los cUMDAs se puede estructurar de dos formas distintas ($1 \times 1 - 20 \times 20$ y $2 \times 2 - 10 \times 10$). Los vecindarios utilizados son C25 y C41 (véase la Figura. 8.3), compuestos por la celda considerada más las 24 celdas más cercanas (para C25) y las 40 (en el caso de C41) medidas con la distancia de Manhattan. El método de selección utilizado es el truncamiento (con $\tau = 0.5$), aplicado en el conjunto de cadenas del vecindario de la sub-población local. Finalmente, todos los algoritmos implementan una etapa de mutación para introducir diversidad en la población. Aunque no es habitual en los EDAs, hemos aplicado esta mutación debido a los buenos resultados que hemos obtenido en experimentos preliminares. La mutación propuesta consiste en cambiarle el valor a los genes de los nuevos individuos con una probabilidad dada.

Para concluir esta sección incluimos un breve estudio del comportamiento teórico de todos los algoritmos propuestos en términos de su presión de selección (diríjase al Capítulo 5 para más detalle).

Tabla 8.1: Parametrización utilizada en todos los algoritmos.

Tamaño de la Población	400 individuos
Selección de los Padres	Selección por Truncamiento, $\tau = 0.5$
Mutación	Bit-flip, $p_m = 1/n$
Reemplazo	Reemp_si_Mejor
Condición de Parada	Óptimo alcanzado o 400000 evaluaciones de fitness
Algoritmos Miembros	MUDA
Caso Celular	*Forma del Vecindario:* $C25$ y $C41$
	Forma de la Población: $1 \times 1 - 20 \times 20$ y $2 \times 2 - 10 \times 10$

Para obtener el tiempo de *takeover* en los EDAs, calculamos la proporción de mejores individuos en la población (tras el truncamiento) y la usamos para generar la nueva población con elitismo (se mantienen los mejores individuos).

En la Figura 8.4 dibujamos las curvas de crecimiento de un MUDA y de cuatro cUMDAs. Para obtener resultados significativos, hemos ampliado la población para este estudio a 64000 individuos. Como se puede observar, a MUDA le corresponde una mayor presión de selección (menor tiempo de *takeover*). Con respecto a los cUMDAs, los dos algoritmos que tienen un individuo en cada celda son los que tienen menor presión de selección (mayor tiempo de *takeover*). La razón es que en los cUMDAs con cuatro individuos en cada celda la capacidad de exploración del algoritmo se penaliza al multiplicarse por cuatro el número de individuos en el vecindario.

8.3.2. Resultados

En esta sección presentamos y analizamos los resultados que obtuvimos en los experimentos para los cinco problemas seleccionados anteriormente. Como dijimos en la sección anterior, los algoritmos que hemos estudiado para nuestras comparaciones son el MUDA estándar más cuatro versiones distintas de UMDAs celulares. Los cinco algoritmos incluyen un operador de mutación. Los cUMDAs estudiados utilizan el vecindario C25 y una o cuatro cadenas por cada celda (llamados C25-1×1-20×20, y C25-2×2-10×10, respectivamente), y estas dos mismas estructuras de población, pero con el vecindario C41: C41-1×1-20×20 y C41-2×2-10×10.

Para obtener resultados significativamente estadísticos, hemos realizado 100 ejecuciones para cada prueba y hemos calculado el análisis de la varianza (ANOVA) o el test de Kruskal-Wallis para comparar nuestros resultados (dependiendo de si el dato sigue una distribución normal o no, respectivamente). Para estas pruebas estadísticas, consideramos un nivel de significación del 95 % (p-valor por debajo de 0.05). Tanto el MUDA como los cUMDAs están escritos en C++ y ejecutados en un Pentium 4 2.4 GHz bajo Linux con 512 MB de memoria RAM. Nótese que estos algoritmos están escritos en C++, por lo que no están incluidos en JCell.

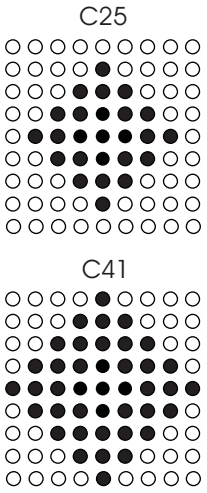


Figura 8.3: Vecindarios utilizados.

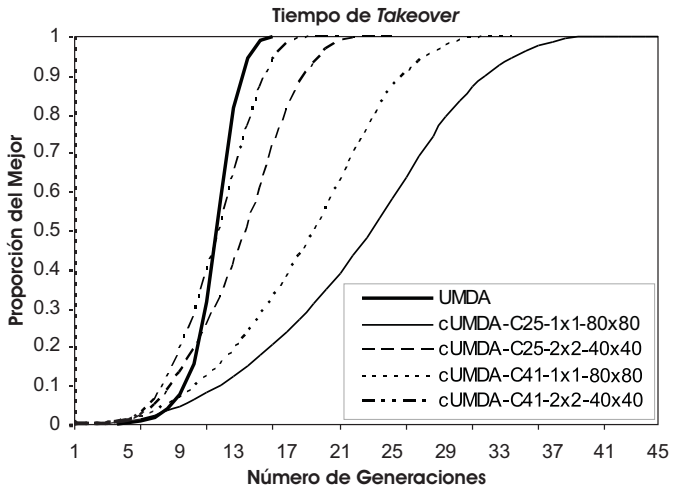


Figura 8.4: Curvas de takeover para los algoritmos estudiados.

Tabla 8.2: Comparación de MUDA y los cuatro cUMDAs propuestos. Tasa de éxito.

Algoritmo	**OneMax**	**Plateau**	**IsoPeak**	**P-PEAKS**	**MTTP**
MUDA	100 %	100 %	0 %	100 %	74 %
cUMDA C25-1×1-20×20	100 %	100 %	100 %	100 %	100 %
cUMDA C25-2×2-10×10	100 %	100 %	100 %	100 %	100 %
cUMDA C41-1×1-20×20	100 %	100 %	100 %	100 %	100 %
cUMDA C41-2×2-10×10	100 %	100 %	100 %	100 %	100 %

Presentamos en la Tabla 8.2 el porcentaje de ejecuciones para los cuales el algoritmo encuentra la solución óptima de los problemas (tasa de éxito o de acierto). Como se puede ver, MUDA tiene dificultades para encontrar el óptimo en el caso del problema MTTP (74 % de las ejecuciones), y no lo encuentra en ninguna ejecución para el problema IsoPeak. Por el contrario, las cuatro versiones celulares estudiadas no tienen ninguna dificultad para obtener el valor óptimo para todos los problemas en cada una de las ejecuciones (tasa de éxito del 100 %).

En la Tabla 8.3 presentamos el número medio de evaluaciones necesario, junto con la desviación estándar, de los cinco algoritmos estudiados para la resolución de los problemas. En la última fila de la tabla se presenta el resultado obtenido en nuestras pruebas estadísticas cuando comparamos todos los algoritmos para cada problema. El símbolo '+' indica la existencia de confianza estadística en la comparación de los cinco algoritmos, es decir, los resultados de al menos dos de los algoritmos comparados son estadísticamente diferentes. Como se puede observar, el algoritmo más eficiente (menor número de evaluaciones) para cada problema (valores en **negrita**) es siempre una de las versiones celulares de MUDA estudiadas.

En la Figura 8.5 mostramos gráficamente los resultados de la Tabla 8.3. Es fácil comprobar que el algoritmo que muestra una menor eficiencia para todos los problemas es MUDA, con la excepción del problema P-PEAKS. Con respecto a las distintas versiones de MUDA celulares, el cUMDA C41-2×2-10×10 es el más eficiente para tres de los cinco problemas estudiados (OneMax, Plateau, y MTTP), aunque es el peor para los otros dos problemas. De todas formas, las diferencias entre los cUMDAs estudiados son, en general, muy bajas (no se encontró confianza estadística en la comparación de los cUMDAs).

Tabla 8.3: Comparación de MUDA y los cuatro cUMDAs propuestos. Evaluaciones de la función de fitness.

Algoritmo	**OneMax**	**Plateau**	**IsoPeak**	**P-PEAKS**	**MTTP**
MUDA	26072.00	18972.00	—	15040.00	169713.51
	±463.20	±1054.86	—	±1448.37	±106624.49
cUMDA C25-1×1-20×20	23298.11	16337.83	176878.71	**13460.97**	14976.44
	±382.76	±584.86	±36384.01	±1650.01	±1348.01
cUMDA C25-2×2-10×10	22037.60	14961.16	218190.00	34994.20	14418.12
	±346.74	±511.65	±47518.06	±13342.22	±1116.66
cUMDA C41-1×1-20×20	22500.89	15454.92	**176138.77**	16915.08	14469.85
	±361.14	±510.05	±41834.80	±3091.56	±981.36
cUMDA C41-2×2-10×10	**21851.72**	**14773.64**	253725.44	41795.32	**14235.52**
	±340.30	±469.24	±58172.59	±15362.06	±1203.63
Test	+	+	+	+	+

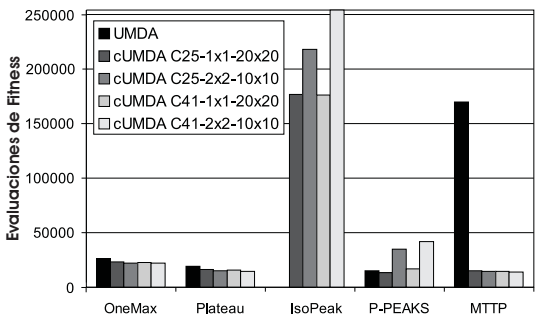


Figura 8.5: Eficiencia de los algoritmos.

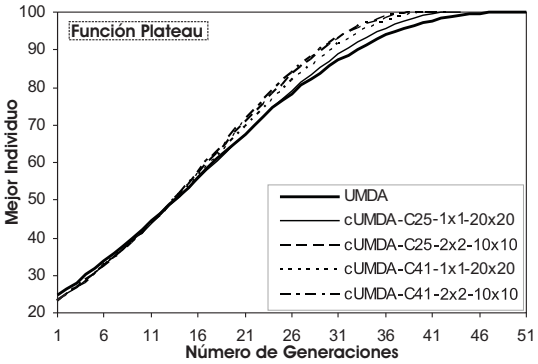


Figura 8.6: Evolución del mejor valor de fitness para Plateau.

Finalmente, en la Figura 8.6 dibujamos un ejemplo de la evolución del mejor valor de fitness para MUDA y los 4 cUMDAs propuestos cuando resolvemos el problema de Plateau. El valor dibujado en cada generación se calcula como la media de 100 ejecuciones. Como podemos comprobar, los dos algoritmos que convergen antes al óptimo son los dos cUMDAs que tienen la población 2×2-10×10, que muestran un comportamiento similar (casi indistinguible), tal y como obtuvimos previamente en el análisis del tiempo de takeover. El algoritmo más lento de todos los estudiados para este problema es MUDA, que encuentra más dificultades que los cUMDAs para evitar caer en óptimos locales y converger al óptimo global debido a su alta presión de selección.

8.4. Conclusiones

En este capítulo hemos investigado un algoritmo de una nueva clase de EDAs descentralizado, llamado EDA celular, basado en el funcionamiento de otros EAs celulares existentes. Nuestra principal motivación ha sido avanzar en el campo de los EDAs estudiando una nueva clase de población descentralizada, fácilmente paralelizable, que ha sido demostrado que obtienen resultados altamente competitivos con otras clases de EAs. Se han probado cuatro aproximaciones basadas en MUDA. La comparación entre los cuatro nuevos cUMDAs y MUDA muestran resultados muy ventajosos para los modelos celulares, ya que MUDA (centralizada) es, en general, el peor algoritmo para todos los problemas, tanto en términos de eficacia (tasa de éxito) como de eficiencia (número de evaluaciones necesarias para alcanzar el óptimo).

La comparación del EDA celular con otros tipos de algoritmos está fuera del alcance de este artículo. Nuestro objetivo no es competir con otros algoritmos existentes, sino presentar un estudio de un nuevo tipo de EDA y analizar su comportamiento frente al equivalente centralizado. Los algoritmos investigados aquí son sólo una primera aproximación y creemos que son susceptibles de una gran mejora después de realizar un ajuste fino de sus parámetros. La comparación entre EDAs y cEDAs se ha llevado a cabo en términos de la presión de selección (en términos teóricos) y de la resolución de un conjunto diverso de problemas discretos (en términos prácticos). En ambos casos, el EDA centralizado es mejorado por los cuatro nuevos modelos celulares (cEDAs) comparados.

Capítulo 9

Diseño de Algoritmos Meméticos Celulares

Los Algoritmos Meméticos (MAs) son técnicas que combinan características de diferentes metaheurísticas, como algoritmos basados en poblaciones y métodos de búsqueda local, y también técnicas de reinicio e intensa hibridación con conocimiento del problema. La principal característica de los MAs es la relación entre la exploración y explotación que aplican sobre el espacio de búsqueda. Nos proponemos en este capítulo sacar provecho en los cGAs de las ventajas que reportan los MAs. Para ello, diseñamos un nuevo modelo de cGA, altamente especializado, que obtenemos tras aplicar en la población de un MA una estructura de vecindario. Al resultado lo llamamos MA celular (cMA). Para evaluar nuestros algoritmos hemos seleccionado el problema de satisfacibilidad (SAT). Éste es un problema combinatorio duro, muy conocido en la literatura, y que tiene aplicaciones prácticas importantes, como diseño [160] e interpretación de imágenes [238], entre otros.

En el campo de la computación evolutiva, los últimos avances muestran claramente que los EAs pueden obtener buenos resultados para SAT cuando son hibridados con otras técnicas, como por ejemplo funciones de fitness adaptativas, operadores específicos, u optimización local [39, 77, 95, 103, 136].

La motivación de este capítulo es estudiar el comportamiento de diferentes cGAs híbridos que tengan funciones de fitness específicas y que incorporen información del problema en los operadores de recombinación, mutación y búsqueda local. Los comparamos con las heurísticas de búsqueda local básica y con dos cGAs canónicos (sin búsqueda local). Además, se han analizado dos formas distintas de incluir la búsqueda local: (i) una búsqueda local ligera computacionalmente aplicada a cada individuo, o (ii) una etapa de búsqueda local de fuerte explotación aplicada con baja probabilidad a los individuos. Aquí realizamos una extensión de [18, 19], incluyendo un estudio preliminar que justifica los algoritmos utilizados, y una comparación de nuestros resultados con otros algoritmos conocidos de la literatura.

Este capítulo se organiza de la siguiente manera. En la Sección 9.1 presentamos nuestra nueva propuesta de cMA. La Sección 9.2 introduce los algoritmos estudiados, incluyendo tres heurísticas sencillas (GRAD, Enfriamiento Simulado –SA– y WSAT), un cGA básico, y varios cMAs, que son el resultado de distintas combinaciones con los tipos de algoritmos mencionados. Nuestros resultados se resumen en la Sección 9.3, y nuestras conclusiones se presentan en la Sección 9.4.

9.1. Algoritmos Meméticos Celulares

Los algoritmos meméticos son algoritmos de búsqueda en los que se utiliza algún conocimiento del problema en uno o más operadores. El objetivo es mejorar el comportamiento del algoritmo original. Los MAs no sólo consideran la búsqueda local, sino también el reinicio, la estructuración y la búsqueda intensa [211]. En este capítulo, nos centramos en el diseño de MAs celulares (cMAs), una nueva clase de algoritmos que introducimos aquí, que son en esencia algoritmos genéticos celulares (cGAs) en los que se incluye parte del conocimiento del problema en la recombinación, mutación, búsqueda local, función de fitness y/o la representación del problema.

En el Algoritmo 9.1 mostramos el pseudocódigo de un cMA canónico. Como se puede observar, la diferencia principal entre los pseudocódigos de los cMA y cGA canónicos es la etapa de búsqueda local incluida en la línea 8 del cMA.

Algoritmo 9.1 Pseudocódigo de un cMA canónico.

```
1.  cMA(cma)        //Parámetros del algoritmo en 'cma'
2.  mientras ! CondiciónParada() hacer
3.    para individuo ← 1 hasta cma.tamañoPobl hacer
4.      vecinos ← CalculaVecindario(cma,posición(individuo));
5.      padres ← Selección(vecinos);
6.      descendiente ← Recombinación(cma.P_c, padres);
7.      descendiente ← Mutación(cma.P_m, descendiente);
8.      descendiente ← BúsquedaLocal(cma.P_LS, descendiente, {intensiva|ligera});
9.      Evaluación(descendiente);
10.     Insertar(posición(individuo), descendiente, cma, poblAuxiliar);
11.   fin para
12.   cma.pop ← poblAuxiliar;
13. fin mientras
```

9.2. Componentes Canónicos y Avanzados en cMAs

En esta sección se da una descripción detallada de los algoritmos utilizados. Específicamente, estudiamos tres heurísticas sencillas para resolver SAT, un cGA básico –el implementado en JCell–, y finalmente los cMAs propuestos.

9.2.1. Tres Técnicas de Búsqueda Local Básica para SAT

En primer lugar, presentamos los tres procedimientos de búsqueda local (LS) que serán utilizados para construir nuestros cMAs, y que también son utilizados independientemente para resolver SAT. Dos de ellos se diseñaron específicamente para este problema: (i) un algoritmo de seguimiento de gradiente (GRAD), basado en la heurística *flip*, desarrollado en esta tesis, y (ii) el conocido algoritmo WSAT. El tercer procedimiento incluido es el Enfriamiento Simulado (SA), una metaheurística de propósito general bien conocida.

GRAD

En esta tesis, hemos desarrollado un nuevo algoritmo de búsqueda local para SAT, llamado GRAD. Se trata de un algoritmo (basado en la heurística flip [136]) que realiza una búsqueda de seguimiento del gradiente en el espacio de soluciones (véase el Algoritmo 9.2). Básicamente, consiste en mutar el valor de una variable en función del número de cláusulas en las que ésta interviene que no se satisfacen con la asignación actual: cuanto mayor sea el número de cláusulas no satisfechas a las que pertenece dicha variable, mayor será la probabilidad de aplicarle la mutación (hacer *flip*). Como se puede observar en el Algoritmo 9.2, se le añade ruido a la búsqueda (con probabilidad 0.2) con el fin de realzar su capacidad de exploración. La principal diferencia de GRAD con la heurística *flip* consiste en que este último cambia el valor de una variable en términos de la ganancia de ese cambio $v = \{v_i | max(\text{sat\_clauses}(\overline{\text{valor}(v_i)}) - \text{sat\_clauses}(\text{valor}(v_i)))\}$ (con $i = 1$ hasta el número de variables, y siendo sat_clauses(valor(v_i)) el número de cláusulas que se satisfacen y que contienen el valor de v_i), mientras que en GRAD el cambio se le aplica a cada variable (v) que satisfaga $v = \{v_i | max(\text{unsat\_clauses}(\text{valor}(v_i)))\}$ con igual probabilidad (independientemente de la ganancia), siendo unsat_clauses(valor(v_i)) el número de cláusulas no satisfechas conteniendo el valor de v_i. Esta diferencia hace que GRAD sea computacionalmente más ligera que la heurística flip.

El funcionamiento de GRAD es sencillo. El algoritmo comienza generando aleatoriamente tanto la mejor solución inicial como el primer individuo (líneas 2 y 4, respectivamente). Después de eso, hasta que se cumpla la condición de parada, se genera repetidamente un nuevo individuo (descendiente) a partir del actual (padre), se evalúa, y reemplaza a la mejor solución actual si es mejor (valor de fitness mayor, puesto que estamos maximizando). El descendiente se crea cambiando los peores genes (variables) del padre –aquellos con el mayor número cláusulas no satisfechas– con igual probabilidad (líneas 9 y 10). Con una probabilidad preestablecida (20 %) se introduce ruido en la búsqueda. En este caso, el descendiente se muta cambiando el valor de una variable escogida aleatoriamente que no satisfaga una o más cláusulas (líneas 6 y 7). Entonces, el proceso de búsqueda se repite para el descendiente (líneas 5 a 14). Cada `MAX_PASOS` iteraciones, la búsqueda se reinicia –el individuo actual se genera aleatoriamente– (línea 4). Hemos establecido el valor `MAX_PASOS` en 10 veces el número de variables. El algoritmo se detiene cuando se encuentra el óptimo (o la mejor solución conocida) o cuando realizamos 2 millones de evaluaciones de la función de fitness (línea 3).

Algoritmo 9.2 Pseudocódigo de GRAD.

```
1.  GRAD(problema)
2.  mejor_ind = NuevoIndividuoAleatorio();
3.  mientras ! CondiciónParada() hacer
4.    ind = NuevoIndividuoAleatorio();
5.    para pasos ← 0 hasta MAX_PASOS hacer
6.      si rand0to1() < prob_ruido entonces
7.        Mutar(ind, VariableAleatoriaQueIntervengaEnAlgunaCláusulaNoSatisfecha());
8.      en otro caso
9.        vars_a_mutar[]=VariablesQueNoIntervienenEnElMáximoNúmeroDeCláusulasSatisfechas();
10.       MutarConIgualProbabilidad(ind,vars_a_mutar);
11.     fin si
12.     Evaluación(ind);
13.     mejor_ind = Mejor(ind,mejor_ind);
14.   fin para
15. fin mientras
```

Algoritmo 9.3 Pseudocódigo de WSAT.

```
1.  WSAT(problema)
2.  mejor_ind = NuevoIndividuoAleatorio();
3.  mientras ! CondiciónParada() hacer
4.    ind = NuevoIndividuoAleatorio();
5.    para pasos ← 0 hasta MAX_PASOS hacer
6.      cláusula = CláusulaNoSatisfechaAleatoria()
7.      si rand0to1() < prob_ruido entonces
8.        Mutar(ind,cláusula[randomEnt(long_cláusula)]);
9.      en otro caso
10.       para i ← 0 hasta long_cláusula hacer
11.         cláusulas_perdidas[i] = CláusulasPerdidasTrasMutar(i);
12.       fin para
13.       Mutar(ind, cláusula[ÍndiceDeMenorValor(cláusulas_perdidas)]);
14.     fin si
15.     Evaluación(ind);
16.     mejor_ind = Mejor(ind, mejor_ind);
17.   fin para
18. fin mientras
```

WSAT

El algoritmo WSAT [253] es una heurística ávida diseñada específicamente para SAT. Básicamente, consiste en seleccionar repetidamente una cláusula no satisfecha (aleatoriamente) y cambiar el valor de una de sus variables (véase el Algoritmo 9.3). Existen muchos métodos para seleccionar esta variable [198]. Entre ellas, hemos adoptado la estrategia *seleccionar mejor*, que consiste en cambiar una variable de la cláusula con una probabilidad dada (prob_ruido = 0.5), y en otro caso cambiar la variable que minimiza el número de cláusulas que son ciertas en el estado actual, pero que podrían convertirse en falsas si se le aplicara el cambio. Después de varios pasos (línea 3), la búsqueda se "reinicia" reemplazando el individuo actual por uno generado aleatoriamente (línea 4). Como en el caso de GRAD, "reiniciamos" cuando el número de pasos es 10 veces el número de variables, y siempre se mantiene almacenada la mejor solución conocida hasta el momento.

SA

El Enfriamiento Simulado [162] es probablemente una de las primeras metaheurísticas con una estrategia explícita para escapar de óptimos locales (véase el Algoritmo 9.4 para el pseudocódigo). La idea principal es permitir algunos movimientos en el espacio de búsqueda que lleven a soluciones de peor calidad con el objetivo de escapar de los óptimos locales. Por eso, se utiliza un parámetro llamado temperatura (`Temp`), de forma que la probabilidad de aceptar un individuo peor que el actual es:

$$p(\text{Temp, offspr, ind}) = e^{\frac{(\text{Get\_Fit(offspr)} - \text{Get\_Fit(ind)}) \cdot \mathbf{10^4}}{\text{targetFitness} \cdot \text{Temp}}} \quad . \tag{9.1}$$

El valor de esta probabilidad disminuye durante la ejecución (línea 19) para reducir la probabilidad de aceptar movimientos con pérdida de calidad de la solución (calculado como se muestra en la Ecuación 9.1) conforme avanza la búsqueda. `Temp` se inicializa a un límite superior dado $\mathtt{T_{max}}$ (línea 5), y se calculan

Algoritmo 9.4 Pseudocódigo de SA.

```
1.  Enfriamiento_Similado(problema, T_max, tasaEnfriamiento)
2.  ind = NuevoIndividuoAleatorio();
3.  mejor_ind = ind;
4.  mientras ! CondiciónParada() hacer
5.    Temp = T_max;
6.    mientras Temp > T_min hacer
7.      descendiente = ind;            // generar un descendiente
8.      para i ← 0 hasta problema.num_vars hacer
9.        si rand0to1() < 1/problema.num_vars entonces
10.         Mutar(descendiente, i);
11.       fin si
12.     fin para
13.     Evaluación(descendiente);
14.     si Fitness(descendiente) >= Fitness(ind) entonces
15.       ind = descendiente;
16.     en otro caso si rando0to1() < p(Temp, descendiente, ind) entonces
17.       ind = descendiente;
18.     fin si
19.     Temp * = tasaEnfriamiento;
20.   fin mientras
21.   mejor_ind = Mejor(descendiente, mejor_ind);
22. fin mientras
```

nuevos individuos mientras que el valor actual de `Temp` sea mayor que un límite inferior dado $\texttt{T}_{\texttt{min}}$ (líneas 6 a 20). El valor de la temperatura desciende en función del parámetro `tasaEnfriamiento`. Si el nuevo individuo es mejor (mayor valor de fitness) que el mejor conocido hasta el momento se acepta como el nuevo mejor individuo (líneas 14 y 15) y, si es peor, lo reemplazará con una probabilidad dada. Después de algunas pruebas experimentales, hemos establecido los valores en $\texttt{T}_{\texttt{max}} = 10$, $\texttt{T}_{\texttt{min}} = 1$, y $\texttt{tasaEnfriamiento} = 0.8$.

9.2.2. Algoritmos Meméticos Celulares Para SAT

En esta sección, proponemos algunos cMAs para el problema SAT. Todos han sido obtenidos hibridando el cGA básico implementado en JCell con diferentes combinaciones de operadores de recombinación general y específica, de mutación y de búsqueda local (ver Tabla 9.1), y también una función de fitness adaptativo especialmente diseñado para SAT, llamada SAW (definida en el Apéndice A).

Tabla 9.1: Operadores utilizados en JCell para resolver SAT.

Operador	**Genérico**	**Específico**
Recombinación	DPX	UCR
Mutación	BM	UCM
Búsqueda Local	SA	GRAD WSAT

Algoritmo 9.5 Pseudocódigo de UCM.

```
1. UCM(Indiv, Ruido)
2. si rand0to1() ≤ Ruido entonces
3.    Mutar(Indiv, randomInt(Indiv.longitud));
4. en otro caso
5.    M_B(Indiv);
6. fin si
```

Utilizamos dos operadores genéticos específicamente diseñados para SAT para la recombinación y la mutación denominados *recombinación de cláusulas no satisfechas* (UCR) y *mutación de cláusulas no satisfechas* (UCM), respectivamente. Los dos operadores se centran en mantener constantes los valores de las variables que satisfacen todas las cláusulas a las que pertenecen. Nuestro UCR es exactamente el mismo operador que el propuesto en [137] (C_B), y UCM es el resultado de añadir ruido a M_B (también propuesto en [137]), como se puede ver en el Algoritmo 9.5. Utilizamos dos operadores genéricos de recombinación y mutación muy conocidos: recombinación de dos puntos (DPX) y mutación binaria (BM).

Las tres heurísticas propuestas anteriormente se han adoptado como operadores de búsqueda local en JCell. Como se puede observar en la Sección 9.3, hemos estudiado algunas configuraciones distintas de estos métodos de búsqueda local. Estas configuraciones difieren en la probabilidad de aplicar el operador de búsqueda local a los individuos y en la intensidad de la búsqueda local. La idea es regular el esfuerzo computacional global para resolver el problema en tiempos asequibles con ordenadores domésticos.

9.3. Análisis Computacional

En esta sección analizamos los resultados de nuestras pruebas para las 12 instancias duras (de $n = 30$ a 100 variables) que componen el conjunto 1 de la batería de problemas propuestos en [39]. Estas instancias pertenecen a la fase de transición de dificultad de SAT, donde se encuentran las instancias más duras, ya que verifican que $m = 4.3 * n$ [209] (siendo m el número de cláusulas).

En las próximas secciones presentamos los resultados que obtenemos en los estudios de este capítulo. Todos los algoritmos se han probado en términos de eficiencia –número medio de evaluaciones necesarias para encontrar una solución (NME)– y eficacia –tasa de éxito (TE)– (véanse las Tablas 9.2, 9.3, 9.5 y 9.6, y la Figura 9.1). Los resultados se han obtenido haciendo 50 ejecuciones independientes de los algoritmos para cada instancia. Hemos aplicado tests estadísticos a nuestros resultados para comprobar si las diferencias son estadísticamente significativas. Se considera un nivel de confianza del 95 %, y las diferencias estadísticamente significativas se muestran con el símbolo '+' ('•' significa que no hay relevancia estadística) en las Tablas 9.2, 9.3, 9.5 y 9.6.

9.3.1. Los Efectos de Estructurar la Población y Utilizar una Función de Adecuación con Pesos Adaptativos (SAW)

En esta sección vamos a justificar la elección de la población estructurada (celular) y el uso de la adaptación paso a paso de funciones ponderadas (SAW) en nuestros cMAs. Con este propósito, estudiamos el comportamiento de JCell.DPX_BM (un cGA que utiliza SAW, y que implementa operadores genéricos de recombinación y mutación –DPX y BM–, respectivamente), comparándolo con su versión generacional –de población no estructurada– (genGA), y JCell.DPX_BM utilizando la función de fitness más clásica para SAT (JCell.DPX_BM_cl), que consiste en contar en número de cláusulas satisfechas por la solución potencial.

Tabla 9.2: Los efectos de estructurar la población y utilizar SAW. Media de evaluaciones para la solución.

Instancia tamaño (n)	**#**	**genGA**	**JCell.DPX_BM_cl**	**JCell.DPX_BM**	**Test**
30	1	49801.0	**3415.7**	7438.0	+
30	2	1135843.2	—	**502208.4**	+
30	3	440886.2	**3024.0**	80029.4	+
40	4	855286.3	31932.0	**13829.8**	+
40	5	66193.9	**4861.4**	9391.7	+
40	6	1603008.0	—	**519868.8**	+
50	7	473839.4	14356.8	**13081.0**	+
50	8	1076077.4	**84088.8**	95379.8	+
50	9	1333872.0	—	**524164.1**	+
100	10	—	—	**601488.0**	+
100	11	—	223310.8	**165484.8**	+
100	12	—	**245232.0**	392871.8	+

Mostramos nuestros resultados en la Tabla 9.2, donde damos para cada instancia, su tamaño, su identificador (#), y el número medio de evaluaciones necesarias para encontrar una solución óptima (NME) de los tres algoritmos (los mejores valores están en **negrita**). Además, el número de ejecuciones en los que se encuentra solución (tasa de éxito) aparece en la Figura 9.1. El criterio de parada consiste en encontrar una solución óptima al problema o realizar un máximo de 100000 generaciones Después de aplicarle el análisis de ANOVA a los resultados de la Tabla 9.2, obtuvimos que hay diferencias estadísticamente significativas en todas las instancias. Se puede ver que genGA es el algoritmo que tiene peores resultados de NME, y JCell.DPX_BM es peor que JCell.DPX_BM_cl con significancia estadística sólo en dos instancias (números 1 y 5). En términos de TE (véase la Figura 9.1), JCell.DPX_BM es mejor que los otros dos algoritmos (mayor eficacia) para las 12 instancias. Es más, sólo JCell.DPX_BM es capaz de encontrar la solución óptima para todas las instancias. Por eso, a partir de estos resultados podemos concluir que JCell.DPX_BM, el algoritmo con población estructurada y que utiliza SAW, es el mejor de los tres comparados, tanto en términos de eficacia (TE) como de eficiencia (NME).

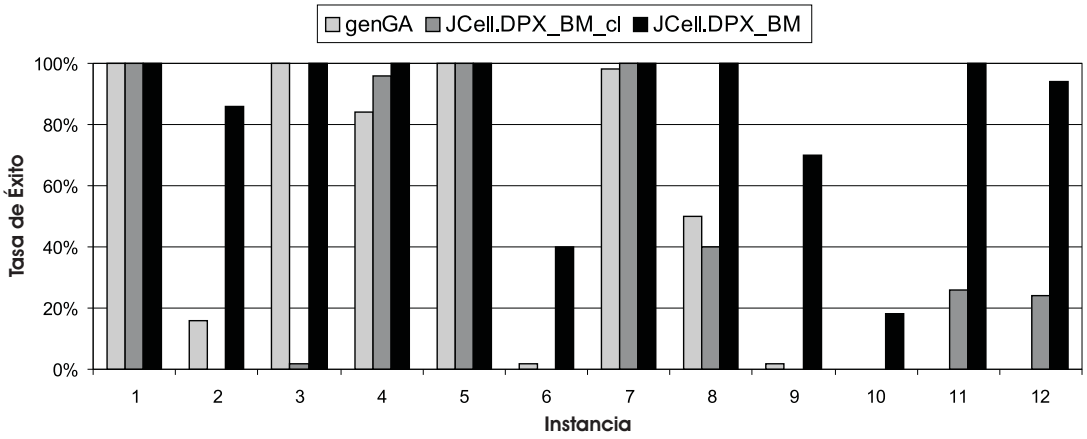


Figura 9.1: Los efectos de estructurar la población y utilizar SAW. Tasa de éxito (TE).

Tabla 9.3: Algoritmos básicos.

Inst.	GRAD		SA		WSAT			JCell.DPX_BM		JCell.UCR_UCM		
	TE	NME	TE	NME	TE	NME		TE	NME	TE	NME	
1	1.00	203.56 ±219.73	1.00	685.22 ±844.74	1.00	**143.64** ±120.32	+	1.00	7438.04 ±3234.60	1.00	104100.48 ±121339.96	+
2	1.00	9681.06 ±9483.55	1.00	63346.60 ±93625.68	1.00	**8575.66** ±9244.08	+	0.86	502208.37 ±491034.68	0.10	697564.80 ±663741.62	•
3	1.00	8520.38 ±7724.11	1.00	16833.44 ±11002.84	1.00	**3984.34** ±4112.95	+	1.00	80029.44 ±54664.78	0.98	269282.94 ±223859.24	+
4	1.00	619.94 ±584.88	1.00	2173.62 ±2076.57	1.00	**199.56** ±193.56	+	1.00	13829.76 ±7801.37	0.10	1364688.00 ±500365.96	+
5	1.00	324.46 ±332.19	1.00	1202.86 ±1045.82	1.00	**103.66** ±88.02	+	1.00	9391.68 ±2478.37	1.00	249137.28 ±236218.19	+
6	1.00	**14368.98** ±13954.02	0.86	271701.47 ±418129.55	1.00	14621.04 ±18617.88	+	0.40	519868.80 ±552312.72	0.00	—	—
7	1.00	496.58 ±359.60	1.00	1614.76 ±1252.34	1.00	**200.84** ±154.81	+	1.00	13080.96 ±3346.94	0.10	1005494.40 ±721439.61	+
8	1.00	1761.74 ±1989.06	1.00	9512.84 ±10226.14	1.00	**793.38** ±870.94	+	1.00	95379.84 ±125768.68	0.00	—	—
9	1.00	82004.84 ±63217.93	1.00	201612.46 ±266218.97	1.00	**77696.42** ±75769.23	+	0.70	524164.11 ±432005.51	0.00	—	—
10	0.94	726522.51 ±525423.23	0.84	510006.12 ±419781.41	1.00	**189785.14** ±198738.78	+	0.18	601488.00 ±364655.49	0.00	—	—
11	1.00	5508.26 ±5940.96	1.00	18123.00 ±20635.35	1.00	**1501.74** ±1264.80	+	1.00	165484.80 ±190927.59	0.00	—	—
12	1.00	8920.38 ±9111.02	1.00	25539.84 ±22393.45	1.00	**1388.92** ±1308.27	+	0.94	392871.83 ±443791.69	0.00	—	—

A partir de la Tabla 9.2 y la Figura 9.1, se puede destacar que el cGA probado, mejora notablemente sus resultados cuando utiliza la función de fitness SAW. Debido a esto, todos los algoritmos estudiados en las siguientes secciones se han implementado utilizando SAW, aunque el uso de SAW no tiene por qué mejorar necesariamente el rendimiento en otros algoritmos, la función de fitness SAW se ha implementado en GRAD, SA y WSAT como se sugiere también en [18]. Se hace esto para poder hacer comparaciones imparciales con los cMAs.

9.3.2. Resultados: Heurísticas no Meméticas para SAT

En esta sección vamos a estudiar el comportamiento de GRAD, SA y WSAT. Además, vamos a probar el comportamiento de dos cGAs diferentes sin búsqueda local: JCell.DPX_BM (ya estudiada en la sección anterior), y JCell.UCR_UCM, que es el resultado de hibridar el primero con operadores de recombinación y mutación específicamente diseñados para SAT (UCR y UCM). Los parámetros utilizados son los de la Tabla 9.4, pero en este caso $P_{\text{LS}} = 0.0$ (no hay búsqueda local). En la Tabla 9.3 mostramos los resultados. La primera cuestión que queremos enfatizar es que sólo WSAT es capaz de resolver el problema en todas las ejecuciones para todas las instancias. Por el contrario, en [136] WSAT sólo resuelve el problema en un 80 % de las ejecuciones para el caso de las instancias más grandes (con $n = 100$), por lo que hemos realizado una mejor implementación de WSAT.

Comparando los tres algoritmos básicos de búsqueda local, se puede observar en la Tabla 9.3 que SA obtiene los peores resultados, tanto en términos de eficacia como de eficiencia (con confianza estadística, excepto para la instancia número 10). GRAD es similar a WSAT en eficacia, pero necesita un mayor número de evaluaciones de la función de fitness para encontrar la solución (menor eficiencia), excepto para la instancia 6 (obtenemos diferencias significativas en las instancias 3, 4, 7, 8 y de la 10 a la 12). Por tanto, podemos concluir que WSAT es la mejor de las tres heurísticas para el conjunto de problemas estudiado, seguido de GRAD. También destacamos la superioridad de los algoritmos diseñados específicamente para el problema frente al patrón genérico de búsqueda como SA.

Tabla 9.4: Parametrización general para los cMAs estudiados.

	JCell.DPX_BM+ {GRAD,SA,WSAT}	**JCell.DPX_BM_i+ {GRAD,SA,WSAT}**	**JCell.UCR_UCM+ {GRAD,SA,WSAT}**	**JCell.UCR_UCM_i+ {GRAD,SA,WSAT}**
Búsqueda Local	Ligera $P_{LS} = 1.0$	Intensiva $P_{LS} = 1.0/tamPobl$	Ligera $P_{LS} = 1.0$	Intensiva $P_{LS} = 1.0/tamPobl$
Mutación	Bit-flip ($P_{bf} = 1/n$), $P_m = 1.0$		UCM, $P_m = 1.0$	
Recombinación	DPX, $P_c = 1.0$		UCR, P_c	
Tamaño de Pob.	144 Individuos			
Selección	Individuo Actual + Torneo Binario			
Reemplazo	Reempl_si_no_Peor			
Crit. de Parada	Encontrar una solución o alcanzar 100.000 generaciones			

Con respecto a los dos cGAs, la Tabla 9.3 muestra el resultado opuesto: los cGAs con operadores genéricos mejoran a los que incluyen mutaciones y recombinaciones hechas a medida. Esto queda claro ya que JCell.UCR_UCM tiene un NME mayor que JCell.DPX_BM (con confianza estadística para todas las instancias, excepto para la 2). Además, JCell.UCR_UCM también tiene una tasa de éxito menor (TE) que JCell.DPX_BM en todos los casos. Por otro lado, JCell.UCR_UCM no es capaz de encontrar el óptimo en ninguna de las 50 ejecuciones realizadas para 6 de las 12 instancias. Probablemente, la razón de este pobre comportamiento de JCell.UCR_UCM con respecto a JCell.DPX_BM radica en que tanto UCR como UCM realizan una explotación demasiado intensa del espacio de búsqueda, resultando en una importante y rápida pérdida de diversidad en la población, consiguiendo así que el algoritmo se estanque en un óptimo local.

Como conclusión final, podemos afirmar a partir de la Tabla 9.3 que los resultados de los dos cGAs sin búsqueda local explícita son siempre peores que los de las heurísticas de búsqueda local estudiadas, tanto en términos de eficacia como de eficiencia. Los mejores resultados de la tabla son los obtenidos por WSAT. Como sospechamos que estos resultados están demasiado ligados a las instancias (especialmente a las de "pequeño" tamaño) vamos a ampliar el conjunto de pruebas al final de la próxima sección con instancias más duras.

9.3.3. Resultados: cMAs

En esta sección estudiamos el comportamiento de un gran número de cMAs con distintas parametrizaciones. Como se puede ver en la Tabla 9.4 (donde se dan los detalles de los cMAs) hibridamos los dos cGAs canónicos de la sección anterior con tres métodos de búsqueda local distintos (GRAD, SA y WSAT). Estos métodos de búsqueda local se han aplicado de dos maneras distintas: (i) ejecutando una etapa de intensa búsqueda local ($10 \times n$ evaluaciones de la función de fitness) a un porcentaje de los individuos (llamado método *intensivo*), o (ii) aplicando una etapa de ligera búsqueda local a todos los individuos, que consiste en realizar 20 evaluaciones de la función de fitness (llamado método *de poca intensidad* o método *ligero*).

Los resultados se muestran en las Tablas 9.5 y 9.6. Si los comparamos con los cGAs de la Tabla 9.3, podemos ver que el comportamiento del algoritmo se mejora en general tanto en eficiencia como en eficacia, especialmente en los casos del cMA con GRAD y WSAT. Por tanto, los tres métodos de búsqueda local utilizados (genéricos y específicos) ayudan al algoritmo a escapar de los óptimos locales. Si comparamos los cMAs estudiados en términos del modo en el que se aplica la búsqueda local (intensa o ligera), podemos concluir que en el caso intensivo siempre se obtienen mejores resultados que en el otro cuando se hibrida el algoritmo con una heurística específica (bien GRAD o WSAT). Esto no es del todo cierto cuando utilizamos SA, ya que SA aplicado de manera intensiva sólo mejora al otro caso en 9 de las 24 pruebas. Por tanto, podemos decir que los cGAs hibridados con una búsqueda local especializada tienen un mejor rendimiento que los que utilizan SA (genérico). Todas estas comparaciones son estadísticamente significativas en 58 de los 65 casos en los que todos los cMAs obtuvieron la solución en casi el 100 % de las ejecuciones.

Tabla 9.5: Los resultados para las hibridaciones de JCell.DPX_BM propuestas.

#	JCell.DPX_BM						JCell.DPX_BM_i					
	+ GRAD		+ SA		+ WSAT		+ GRAD		+ SA		+ WSAT	
	TE	NME	TE	NME	TE	NME	TE	NME	TE	NME	TE	NME
1	1.00	3054.2 ±392.2	1.00	29474.5 ±583.4	1.00	2966.1 ±19.2	1.00	1072.8 ±1112.6	1.00	9649.6 ±25809.9	1.00	**569.9** ±302.8
2	1.00	33598.7 ±51766.6	1.00	195397.6 ±295646.3	1.00	32730.4 ±49353.2	1.00	50886.2 ±44167.7	0.90	559464.3 ±437996.2	1.00	**30885.5** ±22768.8
3	1.00	14761.2 ±24935.1	1.00	33005.4 ±6306.3	1.00	**4104.5** ±3325.1	1.00	20385.8 ±20115.7	1.00	255902.5 ±275734.7	1.00	9418.4 ±10239.6
4	1.00	5018.6 ±2397.8	1.00	31618.8 ±152.9	1.00	3972.9 ±1343.8	1.00	2573.4 ±2497.7	1.00	49310.9 ±64714.9	1.00	**794.7** ±693.7
5	1.00	3575.6 ±1131.5	1.00	31052.9 ±282.7	1.00	3008.3 ±8.4	1.00	1586.0 ±1757.9	1.00	13354.0 ±36668.5	1.00	**628.6** ±437.9
6	0.96	181863.6 ±343020.8	0.96	434235.9 ±519011.4	1.00	81966.1 ±114950.0	1.00	94046.4 ±114105.9	0.72	654160.4 ±476411.6	1.00	**41619.4** ±47466.8
7	1.00	5945.8 ±2416.8	1.00	33621.6 ±7313.2	1.00	4822.6 ±1364.9	1.00	2342.6 ±2972.9	1.00	37446.4 ±70165.5	1.00	**850.8** ±527.5
8	1.00	14930.8 ±7644.5	1.00	47688.6 ±15925.1	1.00	7138.3 ±3957.5	1.00	5164.5 ±5786.7	1.00	195816.2 ±155018.9	1.00	**2097.6** ±1886.8
9	0.80	787149.2 ±528237.4	0.50	720491.5 ±597642.6	1.00	600993.9 ±443475.3	0.82	963177.2 ±585320.7	0.34	883967.7 ±633307.9	1.00	**187814.5** ±148264.1
10	0.06	797880.3 ±824831.9	0.04	1209394.0 ±90058.5	0.06	1189559.7 ±374193.7	0.04	1302489.0 ±346149.9	0.10	1363627.4 ±368403.3	0.80	**792051.2** ±491548.4
11	1.00	58591.3 ±18897.3	1.00	1039910.2 ±205127.9	1.00	35571.0 ±9243.6	1.00	12539.8 ±10851.1	1.00	357207.9 ±422288.9	1.00	**2466.3** ±1846.4
12	0.96	70324.9 ±32808.8	0.98	1051351.2 ±174510.4	1.00	45950.2 ±19870.7	1.00	20018.2 ±19674.3	0.98	409492.6 ±425872.3	1.00	**3196.9** ±2938.3

Tabla 9.6: Los resultados para las hibridaciones de JCell.UCR_UCM propuestas.

#	JCell.UCR_UCM						JCell.UCR_UCM_i					
	+ GRAD		+ SA		+ WSAT		+ GRAD		+ SA		+ WSAT	
	TE	NME	TE	NME	TE	NME	TE	NME	TE	NME	TE	NME
1	1.00	2981.2 ±18.9	1.00	29253.2 ±474.8	1.00	2953.1 ±27.8	1.00	1239.1 ±1467.1	1.00	21710.7 ±46248.4	1.00	**748.8** ±404.1
2	1.00	20294.7 ±23868.0	1.00	187088.9 ±281719.9	1.00	**14879.6** ±18766.3	1.00	58842.3 ±62944.9	1.00	686104.1 ±527621.8	1.00	31457.6 ±33033.8
3	1.00	4048.0 ±2832.1	1.00	41269.5 ±58635.5	1.00	**3641.2** ±1861.2	1.00	25086.8 ±24428.4	1.00	280148.0 ±217802.8	1.00	13614.9 ±13134.6
4	1.00	7853.8 ±9207.1	1.00	31527.8 ±187.2	1.00	3472.5 ±1773.7	1.00	2299.6 ±2937.4	1.00	63190.8 ±110063.6	1.00	**779.4** ±408.9
5	1.00	3466.3 ±1781.8	1.00	30893.9 ±246.8	1.00	2976.1 ±19.9	1.00	1193.1 ±1198.5	1.00	18722.7 ±56165.8	1.00	**624.7** ±369.2
6	1.00	379489.9 ±351593.1	1.00	274977.7 ±389332.4	1.00	162737.1 ±180706.5	1.00	86780.6 ±71185.9	0.94	849405.5 ±584901.9	1.00	**57997.9** ±48455.4
7	1.00	7335.1 ±7980.3	1.00	31715.6 ±134.0	1.00	3532.0 ±1807.9	1.00	1639.8 ±2297.5	1.00	96672.3 ±158359.2	1.00	**678.2** ±507.7
8	1.00	82967.7 ±76765.2	1.00	46418.0 ±15867.9	1.00	27090.5 ±30079.4	1.00	6747.4 ±8070.6	1.00	291700.2 ±225526.0	1.00	**1694.4** ±1619.9
9	0.42	1089600.1 ±642627.4	0.64	1365366.8 ±559506.2	0.56	694014.5 ±548185.0	0.92	566331.3 ±476381.3	0.48	1155717.8 ±529793.2	1.00	**305306.2** ±323215.9
10	0.00	—	0.00	—	0.00	—	0.76	885961.2 ±630092.4	0.16	1099241.9 ±768918.5	1.00	**425377.6** ±415069.5
11	0.00	—	0.64	1743364.38 ±190880.9	0.00	—	1.00	10560.4 ±11327.4	0.90	695508.1 ±855309.5	1.00	**2980.8** ±3334.6
12	0.00	—	0.40	1778928.5 ±200497.9	0.00	—	1.00	16623.6 ±18137.9	0.94	504324.6 ±770231.7	1.00	**3949.3** ±4646.0

Vamos a destacar una excepción interesante, el buen comportamiento de JCell.UCR_UCM+SA para las instancias 11 y 12 con respecto a JCell.UCR_UCM hibridado con GRAD y WSAT, ya que los dos cMAs últimos no son capaces de encontrar la solución óptima en ninguna ejecución. Probablemente la razón sea una intensificación demasiado alta de GRAD y WSAT en la población (recordar que todavía están asociados a UCR y UCM), conduciendo al algoritmo hacia un óptimo local rápidamente.

Tabla 9.7: Eficacia (TE) de diferentes algoritmos evaluados en SAT.

Algoritmo	**n = 30**	**n = 40**	**n = 50**	**n = 100**
SAWEA	1.00	0.93	0.85	0.72
RFEA2	1.00	1.00	1.00	0.99
RFEA2+	1.00	1.00	1.00	0.97
FlipGA	1.00	1.00	1.00	0.87
ASAP	1.00	1.00	1.00	1.00
WSAT [136]	1.00	1.00	1.00	0.80
GRAD	1.00	1.00	1.00	0.98
WSAT	1.00	1.00	1.00	1.00
JCell.DPX_BM_i+WSAT	1.00	1.00	1.00	0.93
JCell.UCR_UCM_i+WSAT	1.00	1.00	1.00	1.00

Ahora vamos a comparar el mejor algoritmo de la Tabla 9.3, WSAT, con el mejor de las Tablas 9.5 y 9.6, JCell.UCR_UCM_i+WSAT. Estos dos algoritmos son los mejores de todos los estudiados en términos de eficiencia y eficacia. Los dos algoritmos encuentran la solución óptima en el 100 % de las ejecuciones (TE=1.0 para cada instancia), pero JCell.UCR_UCM_i+WSAT obtiene peores (más altos) resultados que WSAT en términos de NME (con diferencias estadísticamente significativas). Aunque preveíamos una comparación dura con el mejor algoritmo de la literatura (WSAT), no fue así, ya que obtuvimos en nuestras pruebas una precisión similar y sólo una levemente peor eficiencia. Como sospechábamos que esto sólo se cumpliría en las instancias más pequeñas, decidimos probar estos dos algoritmos con instancias más grandes para comprobar si el cMA era capaz de mejorar al algoritmo del estado del arte WSAT en problemas más duros. Por eso, hemos seleccionado las 50 instancias de 150 variables del conjunto 2 de la misma batería de problemas [39] que estudiamos antes. Los resultados con las instancias más grandes mostraron que JCell.UCR_UCM_i+WSAT resolvió el problema al menos una vez (de 50 ejecuciones) en 26 de las 50 instancias que componen el conjunto, mientras que WSAT encontró la solución para esas mismas 26 instancias y 4 más (el óptimo se encontró sólo una vez en estas 4 instancias). Por tanto, WSAT es capaz de encontrar el óptimo en un mayor número de instancias, con una media de tasa de éxito del 38.24 %, que está bastante cerca de 36.52 %, el valor obtenido por JCell.UCR_UCM_i+WSAT.

Además, la solución media encontrada para este conjunto de problemas (la el valor de fitness óptimo es 645 para todas las instancias) es 644.20 para JCell.UCR_UCM_i+WSAT y 643.00 para WSAT, por lo que cMA es más preciso que WSAT para este conjunto de instancias. Finalmente, si calculamos la media de NME para las instancia resueltas (al menos una vez) por los dos algoritmos podemos ver que cMA (NME = 364383.67) es en este caso más eficiente que WSAT (NME = 372162.36). Por tanto, tal y como sospechábamos, el cMA mejora a WSAT para este conjunto de problemas más grande y más difíciles. De hecho, todos estos resultados representan el nuevo estado del arte ya que "nuestro" WSAT mejora los resultados que existían en la literatura.

9.3.4. Comparación con otros Resultados en la Literatura

En esta sección comparamos algunos de nuestros resultados con los algoritmos estudiados en [136], que se probaron con la misma batería de problemas que nosotros utilizamos aquí. La comparación se proporciona en la Tabla 9.7, donde sólo se muestra la eficacia (tasa de éxito) de los algoritmos porque la eficiencia se mide en [136] siguiendo un criterio distinto al número medio de evaluaciones de la función de fitness (NME) utilizado en este estudio (en ese trabajo se mide la eficiencia en función del número de cambios de bit para obtener la solución).

Como se puede observar en la Tabla 9.7, sólo ASAP [136], nuestra implementación de WSAT y JCell.UCR_UCM_i+WSAT son capaces de encontrar la solución en cada ejecución para todas las instancias. De todos modos, la mayoría de los algoritmos de la tabla tienen una eficacia muy alta para el conjunto de problemas probado, normalmente con tasas de éxito de más del 90 %.

La diferencia en el comportamiento entre nuestro WSAT y el estudiado en [136] se debe a la probabilidad de ruido y a la condición de parada. Por un lado, utilizamos una probabilidad de ruido del 0.5, mientras que Gottlieb et al. no especifican en [136] el valor que utilizan. Por otro lado, nuestro algoritmo termina cuando se realizan 2 millones de evaluaciones de la función de fitness, mientras que en el caso de Gottlieb et al. el algoritmo termina tras alcanzar 300000 cambios de bits. El número de evaluaciones que realiza nuestro algoritmo es mayor que el número de cambios en la implementación de nuestro WSAT (ya que como usamos SAW, el mejor individuo almacenado debe ser re-evaluado). Para comparar con [136] hemos establecido, en experimentos adicionales, una condición de parada de 300000 evaluaciones (se realizarán menos de 300000 cambios) y los valores de TE obtenidos son 1.0, 1.0, 0.99 y 0.95 para el grupo de instancias con n=30, 40, 50 y 100, respectivamente. Estos valores son todavía mayores (mayor eficacia) que los obtenidos por Gottlieb et al.

9.4. Conclusiones

En este capítulo hemos propuesto varios modelos de cMAs, que han sido comparados sobre el problema 3-SAT. Para ello, hemos analizado el comportamiento de tres métodos de búsqueda local y dos cGAs con diferentes operadores (específicos del problema y genéricos). Se han construido 12 cMAs como resultado de hibridar estos dos cGAs con los 3 métodos de búsqueda local, aplicados con dos parametrizaciones diferentes para reforzar la diversificación o la intensificación. Dos de las búsquedas locales, WSAT y GRAD (ésta desarrollada especialmente en esta tesis), han sido diseñadas específicamente para SAT, mientras que SA es un algoritmo genérico.

Hemos visto que los resultados de los cGAs básicos propuestos (sin búsqueda local) están bastante lejos de los obtenidos por los tres métodos de búsqueda local estudiados. Después de hibridar estos dos cGAs canónicos con una etapa de búsqueda local, los cMAs resultantes mejoran sustancialmente el comportamiento de los cGAs originales. Por eso, la etapa de hibridación ayuda a los cMAs a evitar los óptimos locales en los que los cGAs se estancan. Para las instancias más pequeñas, el mejor de los cMAs probados (JCell.UCR_UCM_i+WSAT) es tan preciso como el mejor de los algoritmos presentes en este capítulo (WSAT), pero un poco menos eficiente.

Después de estos resultados, estudiamos el comportamiento de WSAT y JCell.UCR_UCM_i+WSAT (los dos mejores algoritmos) con un conjunto más duro de instancias más grandes. Los resultados confirman nuestras sospechas, ya que cMA es más eficiente que WSAT para estas duras instancias. De todos modos, nuestros resultados contrastan con los de Gottlieb et al., que concluye en [136] que "una investigación experimental preliminar de los EAs para problemas de satisfacción de restricciones, usando tanto una función adaptativa de fitness (basada en f_{SAW}) como búsqueda local, indica que esta combinación no es beneficiosa". Encontramos que esta afirmación no se sostiene cuando aplicamos nuestros algoritmos meméticos estructurados sobre las instancias grandes estudiadas.

Capítulo 10

Diseño de Algoritmos Genéticos Celulares Paralelos

En el campo de los EAs paralelos [8, 51, 220, 267], existe un gran número de implementaciones y algoritmos. La razón de este éxito se debe, en primer lugar, al hecho de que los EAs son inherentemente paralelos, ya que la mayoría de los operadores de variación se pueden ejecutar fácilmente de forma paralela y, en segundo lugar, a que utilizar EAs paralelos nos puede conducir no sólo a un algoritmo más rápido, sino también a un rendimiento numérico superior [29, 130]. Esto es debido a que, como ya hemos comentado varias veces, los EAs paralelos se caracterizan por el uso de una población estructurada (una distribución espacial de los individuos), bien en forma de islas [269] (EAs distribuidos) o bien en una malla de difusión [195] (EAs celulares). Como consecuencia, muchos autores utilizan modelos de poblaciones estructuradas en ordenadores secuenciales (sin paralelismo), y obtienen aún mejores resultados que con EAs secuenciales tradicionales [130].

Los cEAs surgieron como un modelo algorítmico para aprovechar las características de los computadores masivamente paralelos, que poseían cientos o miles de procesadores en paralelo con memoria compartida. En este tipo de computador se hace evolucionar un algoritmo evolutivo, usualmente con una población de un sólo individuo, de forma independiente en cada uno de los procesadores. A cada procesador únicamente se le permitirá interaccionar con sus vecinos con el fin de evitar una posible sobrecarga en las comunicaciones. Este tipo de computador ha perdido popularidad en los últimos años, pero el modelo se ha aplicado satisfactoriamente en máquinas secuenciales tradicionales.

Existen muchas versiones diferentes de cEAs paralelos, tanto en computadoras masivamente paralelas como en clusters de ordenadores con memoria independiente, como se muestra en el Capítulo 3. Todos ellos implementan el mismo modelo algorítmico: el modelo celular. En este capítulo presentamos dos nuevos modelos algorítmicos paralelos en los se mezclan los modelos de grano grueso (modelo de islas) y de grano fino (modelo celular).

En la Sección 10.1 presentamos un nuevo modelo algorítmico, que hemos llamado modelo meta-celular, en el que la población se divide en un conjunto de islas, dispuestas en los nodos de una rejilla toroidal, y dentro de cada una de las cuales se ejecuta un cGA. En la Sección 10.2 adoptamos un modelo presentado originalmente en [7] (llamado dcGA), que es un modelo de islas distribuidas en forma de anillo, dentro de las cuales se ejecutan cGAs. Este modelo será ejecutado en este trabajo sobre una plataforma *grid* compuesta de 125 ordenadores para resolver instancias muy grandes del problema VRP. Finalmente, en la Sección 10.3 resumimos nuestras principales conclusiones.

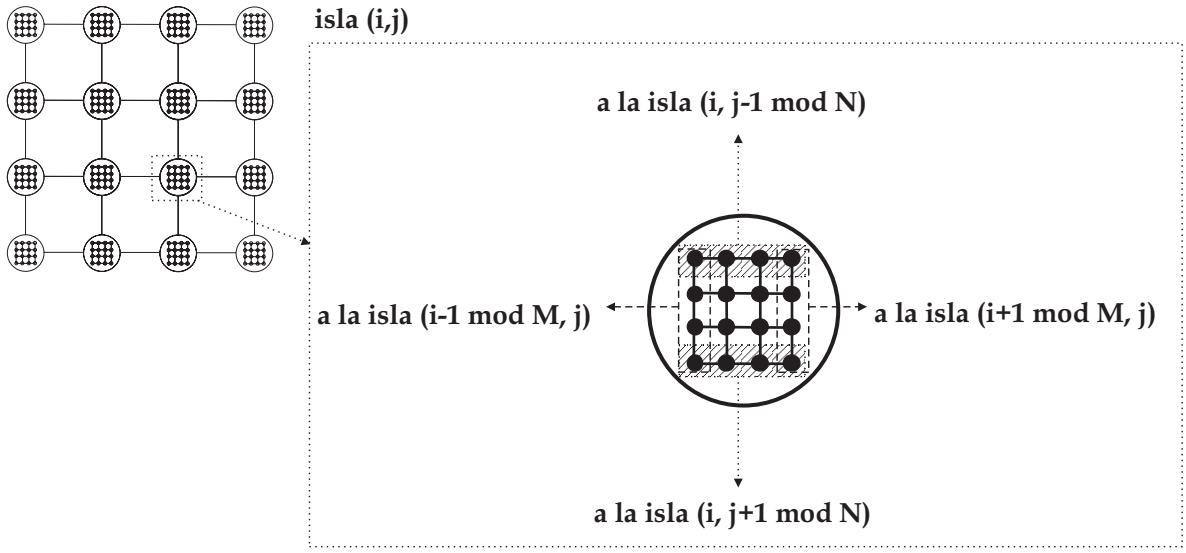


Figura 10.1: Esquema paralelo distribuido del algoritmo genético meta-celular.

10.1. Algoritmo Genético Meta-Celular

Presentamos en esta sección una implementación de un cGA paralelo para sistemas distribuidos, llamado *algoritmo genético meta-celular* (o meta-cGA). En este algoritmo, la población se divide entre todos los procesadores disponibles, aunque el comportamiento global de este cGA paralelo es el mismo que el del celular secuencial. Al principio de cada iteración, todos los procesadores envían los individuos en los extremos norte, sur, este y oeste de su rejilla a sus islas vecinas situadas al norte, sur, este y oeste, respectivamente (como se muestra en la Figura 10.1). La implementación de los algoritmos celulares que se ejecutan dentro de cada isla siguen el modelo canónico. Comparamos aquí el algoritmo con un GA secuencial y varios modelos paralelos.

10.1.1. Parametrización

En esta sección se proporciona una descripción de los parámetros de los algoritmos utilizados. No se ha hecho ningún estudio especial para establecer los parámetros de los algoritmos, por lo que probablemente los algoritmos son susceptibles de mejora. La población completa está formada por 800 individuos. En las implementaciones paralelas, cada procesador tendrá una población de $800/n$ individuos, siendo n el número de procesadores. Todos los algoritmos utilizan el operador de recombinación de un punto –SPX– (con probabilidad 0.7) y el operador de mutación bit-flip (con probabilidad 0.2). En los GAs distribuidos, la migración ocurre en forma de anillo unidireccional, que consiste en que cada subpoblación envía un único individuo (elegido aleatoriamente) a una de sus dos subpoblaciones vecinas. Este individuo es incorporado en la población destino sólo si es mejor que su peor individuo. El paso de migración se realiza cada 20 iteraciones en todas las islas de forma asíncrona. Los experimentos se desarrollaron en 16 PCs Pentium 4 a 2.8 GHz unidos por una red de comunicación *Fast Ethernet*. Debido a la naturaleza estocástica de los algoritmos, realizamos 100 ejecuciones independientes de cada prueba con el fin de producir suficientes datos experimentales para poder obtener resultados estadísticamente significativos.

Tabla 10.1: Media de los resultados para todos los GAs paralelos.

Alg.	**% éxito**	**# evals**	**tiempo**	**Alg.**	**% éxito**	**# evals**	**tiempo**
Sec.	60 %	97671	19.12				
MS2	61 %	95832	16.63	idGA2	41 %	92133	9.46
MS4	60 %	101821	14.17	idGA4	20 %	89730	5.17
MS8	58 %	99124	13.64	idGA8	7 %	91264	2.49
MS16	62 %	96875	12.15	idGA16	0 %	-	-
dGA2	72 %	86133	9.87	meta-cGA2	**85 %**	92286	10.40
dGA4	73 %	88200	5.22	meta-cGA4	83 %	94187	5.79
dGA8	72 %	**85993**	2.58	meta-cGA8	83 %	92488	2.94
dGA16	68 %	93180	**1.30**	meta-cGA16	84 %	91280	1.64

10.1.2. Presentación y Análisis de los Resultados

Compararemos a continuación el algoritmo con algunas otras implementaciones paralelas de GAs sobre las instancias del problema SAT (cuyos detalles se pueden ver en el Apéndice A) propuestas en [156]. Todas estas instancias de SAT están compuestas por 100 variables y 430 cláusulas, por lo que pertenecen a la fase de transición de dificultad del problema SAT (definida en el Apéndice A).

Comencemos nuestro análisis presentando en la Tabla 10.1 el número de ejecuciones que encontraron el valor óptimo (% éxito), el número de evaluaciones (# evals) y el tiempo de ejecución en segundos para todos los algoritmos: un GA secuencial (Sec.), un GA maestro-esclavo (MS), un GA distribuido aislado (idGA, en el que las islas no colaboran entre sí), un GA distribuido en el que las islas sí cooperan (dGA), y el cGA paralelo propuesto (meta-cGA). Los algoritmos paralelos se han probado sobre 2, 4, 8, y 16 procesadores. También mostramos en la Tabla 10.2 el *speedup* de estos algoritmos. Utilizamos la definición *débil* (*Weak*) de *speedup* [27], es decir, comparamos el tiempo de ejecución del modelo paralelo con respecto al secuencial. Los mejores valores de cada tabla están marcados en **negrita**.

Si interpretamos los resultados de la Tabla 10.1 se desprenden varias observaciones. En primer lugar analizaremos cada modelo paralelo. Como se esperaba, el comportamiento de los algoritmos maestro-esclavo es similar a la versión secuencial ya que obtienen el óptimo en el mismo número de intentos y muestrean un número similar de soluciones del espacio de búsqueda. Además, los modelos maestro-esclavo invierten un menor tiempo en encontrar el óptimo, pero el beneficio es muy pequeño para cualquier número de procesadores (Tabla 10.2). Esto es debido a que el tiempo de ejecución de la función de fitness no compensa la sobrecarga en las comunicaciones.

El modelo idGA nos permite reducir el tiempo de búsqueda y obtiene un *speedup* muy bueno (como se puede ver en la Tabla 10.2), casi lineal, pero los resultados son peores que los de los algoritmos secuenciales ya que el número de ejecuciones en las que se encuentra el óptimo es menor, e incluso con 16 procesadores no puede encontrar la solución en ninguna ejecución. Esto no es sorprendente, porque al incrementar el número de procesadores el tamaño de las sub-poblaciones se decrementa, y el algoritmo no es capaz de mantener la suficiente diversidad para encontrar la solución global. Este modelo es el más rápido porque al no producirse migraciones no se producen comunicaciones.

Los GAs distribuidos son mejores que los secuenciales tanto numéricamente como en términos del tiempo de búsqueda ya que obtienen una tasa de éxito más elevada con un número menor de evaluaciones, reduciendo también el tiempo de búsqueda. El *speedup* es bastante bueno pero siempre sublineal y se aleja ligeramente del *speedup* lineal cuando el número de CPUs se incrementa. Esto es, cuando incrementamos el número de CPUs, obtenemos una pequeña pérdida de eficiencia.

Tabla 10.2: *Speedup* débil (*Weak*) de los algoritmos comparados.

Alg.	**Speedup**			
	n = 2	**n = 4**	**n = 8**	**n = 16**
MSn	1.14	1.34	1.40	1.57
idGAn	**2.02**	**3.69**	**7.67**	-
dGAn	1.93	3.66	7.41	**14.7**
cGAn	1.83	3.30	6.50	11.65

Hablando numéricamente, los cGAs son los mejores de los algoritmos comparados ya que obtienen el mejor porcentaje de éxito. Sorprendentemente, obtienen tiempos de ejecución muy bajos, que son sólo ligeramente peores que los de los dGAs, que realizan un número de intercambios significativamente menor.

10.2. Algoritmo Genético Distribuido Celular

En esta sección proponemos un nuevo algoritmo genético celular (cGA) paralelo, que hemos llamado PEGA (Parallel cEllular Genetic Algorithm). En PEGA, adoptamos las dos principales estrategias de paralelización existentes: grano grueso y grano fino. Por tanto, PEGA es un algoritmo paralelo en el que la población está estructurada en dos niveles y de dos formas distintas. Como se puede ver en la Figura 10.2, la población está, en un primer nivel, descentralizada en un modelo de islas que se encuentran colocadas en forma de anillo, de manera que cada subpoblación sólo puede intercambiar información con una subpoblación próxima (migración). En un segundo nivel, con el fin de mejorar la capacidad de búsqueda de las islas, cada isla tiene implementado un cGA, por lo que la población de cada isla está descentralizada en forma de grano fino. Este modelo y sus implementaciones paralelas en clusters de máquinas distribuidas se propusieron inicialmente en [7]. Cada uno de los cGAs que se ejecutan en las islas es un proceso maestro que envía a otros ordenadores esclavos las tareas más costosas que se realizan en cada individuo.

PEGA [88] ofrece la posibilidad de ser ejecutado en plataformas *grid* gracias a que las comunicaciones han sido implementadas utilizando ProActive [3]. ProActive es una biblioteca de Java para la programación en entornos *grid* de una forma simple. ProActive proporciona mecanismos para la creación y manejo en máquinas remotas de objetos que colaboran entre sí (llamados *objetos activos*). El acceso a un objeto activo se realiza como si estuviera en la máquina local, y ProActive se encarga de realizar automáticamente las comunicaciones necesarias utilizando Java RMI.

Como principal contribución de esta sección, hemos diseñado un nuevo algoritmo genético celular híbrido que se ejecuta en modo paralelo sobre plataformas *grid*. El algoritmo se ha utilizado para resolver las instancias más grandes conocidas del problema VRP, presentadas en [181], y se han conseguido mejorar las mejores soluciones conocidas obtenidas por un algoritmo en la mayoría de las instancias estudiadas.

En la Sección 10.2.1 presentamos los parámetros utilizados en el algoritmo, así como las instancias grandes de VRP que pretendemos resolver. En la Sección 10.2.2 presentamos y analizamos los resultados obtenidos.

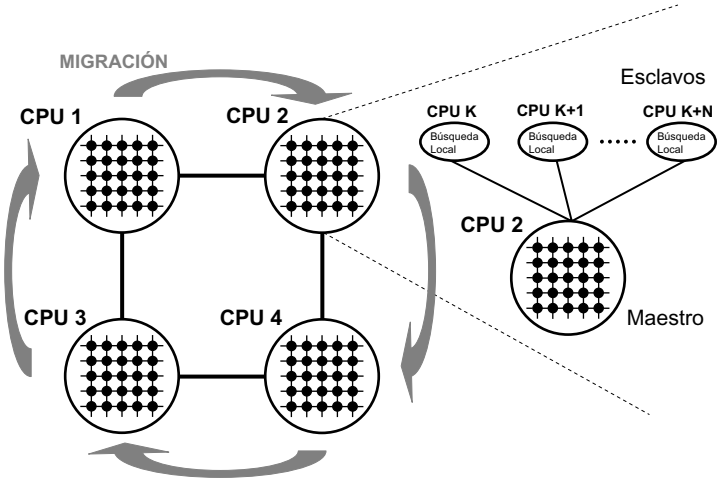


Figura 10.2: Estructura del algoritmo PEGA.

10.2.1. Parametrización

Recientemente, Li, Golden y Wasil presentaron en [181] un nuevo conjunto de instancias del problema CVRP caracterizado fundamentalmente por el elevado número de clientes de que estaban compuestas. Este conjunto de problemas fue llamado VLSVRP, acrónimo de *Very Large Scale VRP*, que se puede traducir como VRP de escala muy grande. El tamaño de las instancias propuestas por los autores en VLSVRP oscila entre los 560 y los 1200 clientes, mientras que los conjuntos de instancias más aceptados y utilizados en la comunidad científica hasta entonces estaban formados por instancias de entre 50 y 199 clientes en el caso de CMT [58], o problemas desde 200 a 483 clientes en el caso de las propuestas en [125] por Golden et al. Algunas características importantes adicionales de VLSVRP es que se ha creado utilizando un generador de instancias, por lo que resulta sencillo obtener instancias de mayor tamaño (para más detalles acerca de este generador de instancias, consulte [181]). Además, las instancias generadas son geométricamente simétricas, siguiendo un patrón circular. Esto nos permite sin mucho esfuerzo obtener una estimación visual de la mejor solución al problema. Todas las instancias de VLSVRP incluyen restricciones en la longitud máxima de las rutas.

Como ya se comentó anteriormente, PEGA está formado por un conjunto de islas, dentro de las cuales se ejecuta un cGA. Para resolver el problema VRP, estos algoritmos son cGAs canónicos que incorporan un mecanismo de búsqueda local similar al empleado en el Capítulo 13. Esta búsqueda local que se le aplica a cada individuo es el trabajo más duro computacionalmente del ciclo reproductivo del cGA, por lo que es la tarea que se envía a los procesadores esclavos. Antes de presentar la parametrización del algoritmo, se detallan a continuación algunos aspectos importantes de la implementación del cGA utilizado:

- **Representación de los individuos.** La representación utilizada para los individuos es la llamada GVR (*Genetic Vehicle Representation*) [232]. En GVR, el genotipo contiene una permutación de enteros (representando a los distintos clientes), así como información de la posición en la que termina cada ruta. En nuestro algoritmo no se admite la existencia de individuos no factibles, por lo que cuando se exceden bien la capacidad del vehículo o bien la longitud máxima en una ruta, se procede a su reparación mediante la partición de la ruta en dos rutas (o más si es necesario) distintas que no violen ninguna restricción.

Algoritmo 10.1 Pseudocódigo del operador de recombinación en PEGA.

// Sean I_1 e I_2 los padres seleccionados del vecindario;
Seleccionar una sub-ruta aleatoria $SR = \{a_1, \ldots, a_n\}$ de I_2
Buscar el cliente $c \notin SR$ más cercano geográficamente a a_1
Eliminar todos los clientes de I_1 que están en SR
El descendiente se obtiene tras insertar SR en el material genético de I_1 de forma que a_1 esté situado inmediatamente después de c

- **Generación de la población inicial.** Los individuos de la población inicial son generados de forma aleatoria y a continuación son modificados particionando las rutas no factibles (como se detalla en [232]) para forzar que representen soluciones válidas al problema (es decir, que no incumplen ninguna restricción).

- **Operador de recombinación.** El operador de recombinación utilizado en PEGA es el *cruce genérico* (*generic crossover*), propuesto originalmente en [232]. Este operador tiene la principal característica de promover de una forma importante la diversidad en la población. Consideramos que ésta es una característica muy importante del operador, ya que en nuestra experiencia en trabajos anteriores [14, 16, 17] la población completa termina convergiendo hacia un mismo óptimo local en multitud de ocasiones. Este operador de cruce es algo inusual, puesto que el descendiente generado no contiene exactamente una mezcla del material genético de los padres, como puede verse en la Figura 10.3. En el Algoritmo 10.1 se muestra un pseudocódigo de este operador de recombinación.

- **Mutación.** La mutación está compuesta por cuatro tipos de mutaciones distintas que se aplican con diferentes probabilidades (sólo se aplica una de ellas en cada caso), igual que en [232]. El uso de estos cuatro operadores de mutación nos permite modificar el itinerario de una ruta, mover clientes entre rutas, y añadir o eliminar rutas, al igual que sucede con el operador de cruce utilizado. Estos operadores son (ver Figura 10.4):

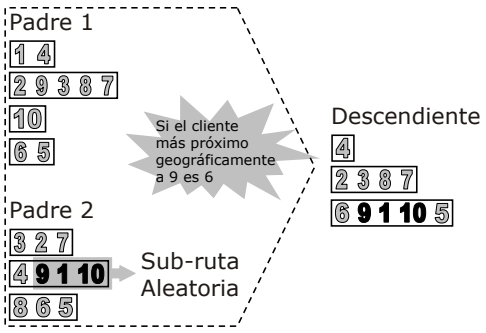


Figura 10.3: El operador de recombinación utilizado: el *cruce genérico*.

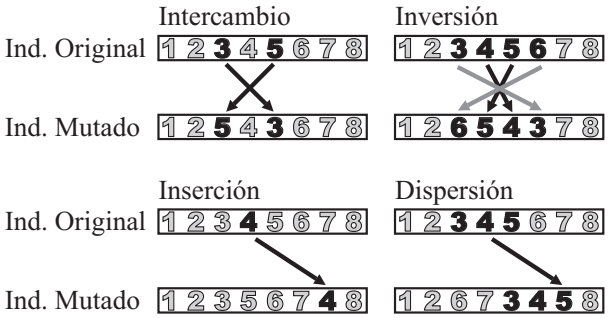


Figura 10.4: Los operadores de mutación.

Tabla 10.3: Parametrización utilizada en PEGA.

Tamaño de Población	Islas: 100 Individuos (10×10) Total: 400 Individuos (4 islas)
Vecindario	NEWS
Selección de Padres	Torneo Binario + Individuo Actual
Recombinación	*Generic Crossover* [232], $p_c = 1.0$
Mutación de Individuos	Intercambio ($p_{\text{int}} = 0.05$), Inversión ($p_{\text{inv}} = 0.1$), Inserción ($p_{\text{ins}} = 0.05$), y Dispersión ($p_{\text{disp}} = 0.15$)
Reemplazo	Reempl_si_Mejor
Búsqueda Local	1-Intercambio + 2-Opt, 50 Pasos de Optimización por Método
Frec. de Migración	Cada 10^4 evaluaciones
Condición de Parada	Alcanzar $5 \cdot 10^5$ Evaluaciones en Cada Isla

  - **Intercambio.** Consiste en intercambiar la posición de dos clientes (pertenecientes a la misma ruta o no) elegidos aleatoriamente.
  - **Inversión.** Invierte el orden de visita de los clientes que se encuentran entre dos clientes elegidos aleatoriamente. En este caso, todos los clientes deben pertenecer a la misma ruta.
  - **Inserción.** Consiste en insertar un cliente seleccionado aleatoriamente en otro lugar del cromosoma del individuo también aleatorio, perteneciente a la misma ruta o a otra.
  - **Dispersión.** Es similar al operador de inserción, pero aplicado a una sub-ruta (conjunto de clientes) en lugar de un único cliente.

- **Búsqueda Local.** El método de búsqueda consiste en aplicar hasta 50 pasos de búsqueda por *1-Intercambio* [226], después se aplican hasta 50 pasos de *2-Opt* [66] a cada ruta de la solución obtenida tras el *1-Intercambio*. El método de *1-Intercambio* consiste en intercambiar un cliente de una ruta por otro perteneciente a otra ruta, o insertar un cliente de una ruta en otra ruta distinta. Por otro lado, 2-Opt trabaja siempre dentro de cada ruta, y consiste en eliminar dos ejes de una ruta y conectar los clientes en la otra forma posible. Como estos dos métodos de búsqueda local son deterministas, la búsqueda para en el caso de que no se haya logrado mejorar la solución en un paso del algoritmo. Esto nos permitirá reducir considerablemente el tiempo de ejecución.

PEGA ha sido diseñado con la idea de ser utilizado para resolver las instancias que forman VLSVRP. Debido a la dificultad y gran tamaño de estos problemas, decidimos ejecutar PEGA en un entorno *grid* compuesto por un máximo de 125 ordenadores de diversas características, como por ejemplo PCs o estaciones de trabajo Sun. El algoritmo está codificado utilizando Java y la biblioteca ProActive para gestionar el entorno *grid*. La parametrización utilizada en PEGA para las pruebas realizadas se detalla en la Tabla 10.3. En concreto, hemos descentralizado el algoritmo en 4 islas en anillo, que intercambian sólo un individuo con una única isla vecina cada 10^4 evaluaciones. En cada una de estas islas se ejecuta un cGA canónico que trabaja sobre una rejilla de 10×10 individuos, y que incorpora un mecanismo de búsqueda local similar al empleado en el Capítulo 13. Uno de los padres es seleccionado mediante torneo binario dentro del vecindario del individuo actual (que está compuesto por los individuos situados al

Tabla 10.4: Resultados obtenidos por PEGA para el conjunto de instancias VLSVRP.

Instancia	Tamaño	MSC	Mejor	Media	Tiempo (h)
VLS21	560	**16212.83**§	**16212.83**	$16212.83_{\pm 4.42e-4}$	10.08
VLS22	600	**14641.64**†	14652.28	$14755.90_{\pm 98.88}$	10.08
VLS23	640	**18801.13**§	**18801.13**	$18801.13_{\pm 4.32e-6}$	11.28
VLS24	720	**21389.43**§	**21389.43**	$21389.43_{\pm 7.63e-6}$	13.20
VLS25	760	**17053.26**§	17340.41	$17423.42_{\pm 72.10}$	18.00
VLS26	800	23977.74§	**23977.73**	$23977.73_{\pm 3.49e-5}$	23.76
VLS27	840	**17651.60**†	18326.92	$18364.57_{\pm 37.66}$	26.40
VLS28	880	**26566.04**§	**26566.04**	$26566.04_{\pm 1.33e-6}$	30.00
VLS29	960	**29154.34**§	**29154.34**	$29154.34_{\pm 4.24e-5}$	39.60
VLS30	1040	**31742.64**§	31743.84	$31747.51_{\pm 3.67}$	48.72
VLS31	1120	**34330.94**§	**34330.94**	$34331.54_{\pm 0.60}$	60.00
VLS32	1200	**36919.24**§	37423.94	$37431.73_{\pm 7.79}$	74.88

§ Solución estimada visualmente [181]; † ORTR [181]

norte, sur, este y oeste –NEWS). El otro padre es el propio individuo actual. Ambos padres se recombinan con una probabilidad del 100 % usando el operador *generic crossover*, y el individuo resultante se muta utilizando los operadores de intercambio, inversión, inserción, y dispersión con probabilidades 0.05, 0.1, 0.05, y 0.15, respectivamente. Posteriormente, se le aplica al descendiente un método de búsqueda local que está compuesto por 50 pasos del algoritmo *1-Intercambio* [226] (explora combinaciones de clientes entre diferentes rutas) y tras esto, otros 50 pasos de *2-Opt* [66] (para optimizar cada una de las rutas generadas por separado). El recién generado descendiente reemplazará al individuo actual en la población sólo si su valor de fitness es mejor.

En la implementación propuesta, la migración se realiza cada 10^4 evaluaciones mediante el intercambio del mejor individuo de cada subpoblación con su subpoblación vecina. Cuando una subpoblación recibe un individuo en la migración, reemplaza a su individuo con peor valor de fitness por este.

10.2.2. Presentación y Análisis de los Resultados

En la Tabla 10.4 presentamos los resultados obtenidos con PEGA para la batería de problemas VLSVRP (el mejor resultado para cada una de las 12 instancias está marcado en **negrita**). En concreto, se muestra el nombre de las instancias estudiadas, su tamaño (número de clientes a visitar), la mejor solución conocida para cada instancia (MSC), la mejor solución encontrada por nuestro algoritmo, la media de las soluciones reportadas por PEGA y el tiempo medio en horas empleado por el algoritmo.

Los valores que se muestran en la Tabla 10.4 han sido obtenidos tras realizar cuatro ejecuciones para las instancias más pequeñas (VLS21 a VLS25) y dos para el resto. Este bajo número de ejecuciones realizadas es debido al elevado tiempo de cómputo requerido por el algoritmo para resolver el problema, que va desde las 10 horas para las instancias más pequeñas hasta las 72 horas para el caso de VLS32. Estos elevados tiempos de cómputo vienen motivados fundamentalmente por dos aspectos: (i) por un lado, utilizamos una búsqueda local pesada, creciendo casi exponencialmente el tiempo necesario para realizarla con el tamaño de la instancia a resolver; (ii) por otro lado, debido a que no está demostrado que se conozca el óptimo para ninguna de las instancias, la condición de terminación del algoritmo depende únicamente del número de evaluaciones realizadas, y este número debe ser lo suficientemente grande como para que el algoritmo tenga tiempo suficiente para encontrar o incluso superar a la mejor solución conocida.

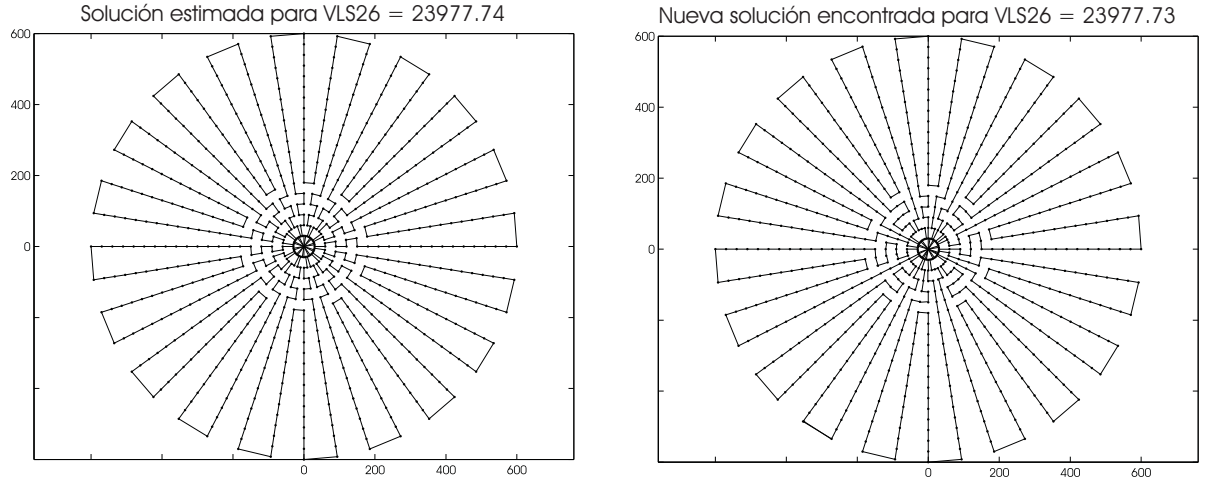


Figura 10.5: Nueva mejor solución para VLS26.

Cabe destacar los buenos resultados obtenidos aún cuando el número de ejecuciones realizadas es tan bajo, ya que PEGA encuentra la mejor solución conocida para 7 de las 12 instancias estudiadas. Incluso mejora la mejor solución conocida hasta ahora para la instancia VLS26, ya que, aunque la diferencia es de únicamente una centésima, representan soluciones bien distintas, como puede observarse en la Figura 10.5. Además, la mejor solución conocida para todas las instancias excepto VLS22 y VLS27 ha sido estimada visualmente utilizando las propiedades geométricas de los problemas, y ningún algoritmo hasta el presente trabajo había sido capaz de encontrarlas. En el caso de VLS22 y VLS27 las soluciones han sido encontradas por distintos algoritmos RTR (*Record-to-Record*) durante las pruebas realizadas para desarrollar VRTR en [181] (ORTR significa *otros RTR*), pero no se facilitan en ese trabajo ni detalles de las implementaciones de estos algoritmos ni referencias en las que se puedan consultar dichos detalles.

En la Figura 10.6 se muestra una comparación de las mejores soluciones obtenidas por PEGA y los tres algoritmos que conforman el estado del arte para VLSVRP. Estos tres algoritmos son distintas parametrizaciones de VRTR, propuesto en [181]. Como se puede ver, PEGA obtiene mejores resultados que los tres algoritmos comparados para todas las instancias excepto VLS25, VLS27 y VLS32. En la Tabla 10.5 mostramos la comparación de los cuatro algoritmos en términos de la diferencia (en porcentaje) entre la solución reportada por los algoritmos para cada problema y la mejor solución conocida para el problema dado (los mejores valores están en **negrita**). También se presentan los valores obtenidos por un cGA maestro esclavo con la misma parametrización que los cGAs utilizados en PEGA, con el fin de mostrar el buen rendimiento del nuevo modelo de paralelización utilizado en PEGA. El valor mostrado en esta tabla se puede entender como una medida de la calidad de los resultados obtenidos por los algoritmos. Se puede ver cómo PEGA es notablemente más robusto que los otros cuatro algoritmos para el conjunto de problemas estudiado, ya que es el algoritmo que más se aproxima, obteniendo los mejores resultados en 9 de los 12 problemas. Además, PEGA encuentra la mejor solución conocida en 7 instancias, soluciones que nunca antes habían sido encontradas por un algoritmo como ya se ha comentado anteriormente.

En la última fila de la Tabla 10.5 se presenta la media de las diferencias entre las soluciones encontradas por cada algoritmo y las mejores soluciones conocidas para las 12 instancias de VLSVRP. Como puede verse, PEGA consigue el mejor valor de los cinco algoritmos comparados, siendo este valor la mitad del resultado del mejor de los algoritmos comparados (VRTR con $\alpha = 1.0$).

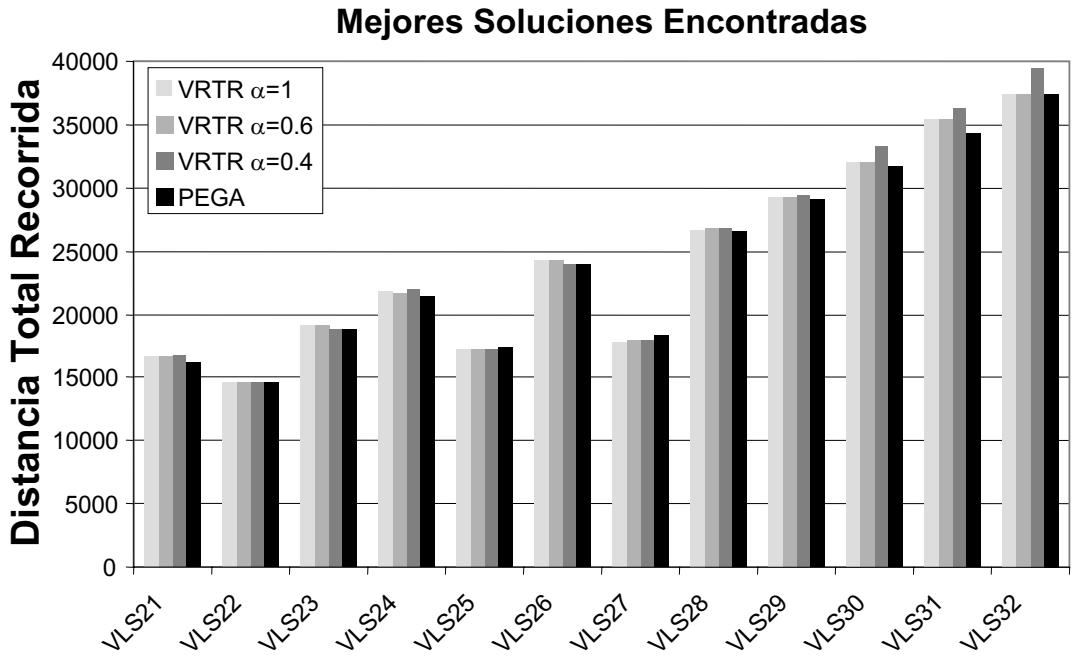


Figura 10.6: PEGA vs. el estado del arte para VLSVRP.

Tabla 10.5: Diferencia (en %) entre la mejor solución conocida y los resultados de PEGA, un cGA MS, y los algoritmos del estado del arte.

Instancia	**VRTR**			**PEGA**	**cGA MS**
	$\alpha = 1.0$	$\alpha = 0.6$	$\alpha = 0.4$		
VLS21	2.41	2.56	3.25	**0.00**	0.02
VLS22	**0.07**	0.10	0.19	**0.07**	2.17
VLS23	1.09	1.56	0.20	**0.00**	**0.00**
VLS24	1.85	1.06	2.54	**0.00**	$5.61e-3$
VLS25	0.58	0.65	**0.55**	1.68	3.53
VLS26	0.88	0.93	0.13	**0.00**	0.05
VLS27	**0.97**	1.61	1.42	3.83	4.83
VLS28	0.15	0.82	0.83	**0.00**	**0.00**
VLS29	0.09	0.10	0.85	**0.00**	0.02
VLS30	0.74	0.69	4.74	**3.78e − 3**	0.64
VLS31	3.02	2.98	5.85	**0.00**	0.35
VLS32	1.36	**1.33**	6.76	1.37	2.02
Media	1.10	1.20	2.28	**0.58**	1.14

10.3. Conclusiones

En este capítulo hemos propuesto dos nuevos modelos de cGAs paralelos. El primero de ellos, el meta-cGA, implementa un modelo similar al de un cGA secuencial. Para ello, crea una sub-población en cada procesador, en el que se ejecuta un cGA independiente. Al final de cada evaluación, cada isla envía a las islas vecinas situadas al norte, sur, este y oeste, los individuos en los extremos norte, sur, este y oeste, respectivamente, de la rejilla del cGA que ejecuta. El algoritmo se ha comparado con un GA secuencial y con los modelos de GA paralelos más comunes en la literatura sobre el problema SAT. Para este problema, hemos encontrado que el modelo maestro-esclavo no es adecuado, ya que la sobrecarga provocada por las comunicaciones no compensa el tiempo de ejecución de la función objetivo. El modelo de islas independientes obtiene siempre tiempos de ejecución muy bajos, pero la calidad de la solución es peor. El uso de los modelos distribuidos y celulares nos ha llevado a una mejora en el porcentaje de soluciones encontradas y en el tiempo de búsqueda. La versión distribuida necesita menos tiempo que la celular ya que realiza un menor número de intercambios, pero el GA celular obtiene las mejores tasas de éxito de todos los algoritmos estudiados.

En segundo lugar, se ha presentado un potente algoritmo para resolver instancias muy duras del problema CVRP. El algoritmo propuesto, PEGA, es un modelo distribuido en islas, en cada una de las cuales se ejecuta un algoritmo genético celular, por lo que la población se encuentra estructurada en dos niveles. Adicionalmente, debido a la complejidad que supone el tipo de problema que estudiamos, este modelo ha sido paralelizado de forma que pueda funcionar sobre plataformas *grid* utilizando la biblioteca de Java ProActive. El modelo de paralelización es maestro/esclavo, y cada una de las islas se ejecuta en una CPU.

PEGA ha sido comparado con los algoritmos del estado del arte para la batería de instancias más grandes existentes para el problema CVRP, la recientemente propuesta VLSVRP. Como resultado, PEGA no sólo ha encontrado los mejores resultados para 9 de las 12 instancias estudiadas, sino que ha encontrado la mejor solución conocida para 7 instancias (nunca antes encontrada por un algoritmo, ya que estas soluciones son cotas estimadas visualmente aprovechando su simetría), y ha encontrado una nueva mejor solución para la instancia VLS26.

Por último, se ha comparado PEGA con un cGA (con la misma parametrización que la de los cGAs utilizados en PEGA) paralelizado siguiendo un modelo maestro esclavo, por lo que el modelo algorítmico es exactamente igual al de un cGA secuencial. Los resultados nos muestran que el cGA maestro esclavo es peor que PEGA para todas las instancias excepto en dos, para los que las dos algoritmos obtienen los mismos resultados.

Capítulo 11

Diseño de Algoritmos Genéticos Celulares para Optimización Multi-objetivo

En la mayoría de los problemas de optimización del mundo real es necesario maximizar o minimizar más de una función. Generalmente, la optimización multi-objetivo no está restringida a encontrar una única solución para un problema de optimización multi-objetivo (MOP) dado, sino un conjunto de soluciones llamado *soluciones no dominadas*. Cada solución de este conjunto se dice que es un *óptimo de Pareto*, y cuando las dibujamos en el espacio objetivo se conocen como *frente de Pareto*. Obtener el frente de Pareto de un MOP dado es la principal meta de la optimización multi-objetivo. En general, el espacio de búsqueda en los MOPs suele ser muy grande, y evaluar las funciones puede llevar mucho tiempo. Estas características hacen que sea difícil aplicar técnicas deterministas, por lo que en este dominio tradicionalmente se han propuesto técnicas estocásticas. Entre ellas, los algoritmos evolutivos (EAs) han sido analizados por muchos investigadores, y algunos de los algoritmos más conocidos para resolver MOPs pertenecen a esta clase (por ejemplo NSGA-II [80], PAES [167], y SPEA2 [301]).

Los EAs son realmente adecuados para tratar los MOPs debido a su habilidad para encontrar múltiples soluciones relacionadas en una única ejecución. Se ha demostrado que los cGAs son muy efectivos para resolver un conjunto diverso de problemas de optimización de un sólo objetivo en escenarios tanto teóricos como del mundo real [15, 20], pero se le ha prestado poca atención a su uso en el campo de la optimización multi-objetivo. En [178], se presenta una estrategia de evolución multi-objetivo siguiendo el modelo depredador-presa. Este es un modelo parecido al de un cGA, ya que las soluciones (presas) se sitúan en los vértices de un grafo conexo no dirigido, definiendo así un vecindario, donde son cazados por los depredadores. Murata y Gen presentaron en [212] un algoritmo en el que, para un MOP con n-objetivos, la población se estructura en un espacio n-dimensional de pesos, y la posición de los individuos (celdas) depende de su vector de pesos. Por tanto, la información dada por el vector de pesos de los individuos se utiliza para guiar la búsqueda. En [163] se presenta un algoritmo evolutivo de metapoblación (denominado MEA). Este algoritmo es un modelo celular con la peculiaridad de que pueden ocurrir desastres en la población, muriendo así todos los individuos situados en la zona del desastre (extinción). Además, los individuos pueden volver a ocupar estas áreas vacías (colonización). Por tanto, este modelo permite un tamaño de la población flexible, combinando las ideas del celular y de las poblaciones dis-

tribuidas espacialmente. Finalmente, en el Capítulo 15 se propone cMOGA [10, 21], el primer algoritmo genético celular multi-objetivo basado en el modelo de cGA canónico, según nuestro conocimiento. En ese capítulo, cMOGA se utiliza para optimizar una estrategia de difusión (o *broadcasting*) específicamente diseñada para redes móviles ad hoc.

Nuestra propuesta en este capítulo se denomina MOCell, y es una versión mejorada de cMOGA, por lo que es también una adaptación de un cGA canónico al campo multi-objetivo. MOCell utiliza un archivo externo para almacenar las soluciones encontradas no dominadas durante la ejecución del algoritmo, como hacen muchos otros algoritmos evolutivos multi-objetivo (por ejemplo, PAES, SPEA2, o cMOGA). De todos modos, la característica principal que diferencia a MOCell con respecto a estos algoritmos es la realimentación de individuos desde el archivo que se realiza después de cada iteración, reemplazando a algunos individuos de la población actual seleccionados de manera aleatoria.

MOCell no está implementado en la biblioteca JCell, en la que sí se encuentra, sin embargo, cMOGA. Está previsto incorporar en cMOGA las características mejoradas que se añadieron a MOCell para próximas versiones de JCell. De cualquier forma, MOCell está disponible públicamente dentro de la biblioteca jMetal [92], que se puede descargar gratuitamente de [91].

Como contribuciones de este capítulo, destacaremos que estamos proponiendo un nuevo cGA para resolver MOPs continuos. Dicho algoritmo utiliza un archivo externo y realimenta algunas soluciones desde el archivo a la población. Evaluamos MOCell utilizando un conjunto de MOPs con y sin restricciones, y lo comparamos con NSGA-II y SPEA2, los dos principales algoritmos del estado del arte para resolver MOPs. Por último, proponemos diferentes estrategias de implementación (en el reemplazo y la realimentación de individuos) para mejorar el comportamiento de MOCell, comparando de nuevo la mejor de estas versiones con NSGA-II y SPEA2.

El resto del capítulo se organiza de la siguiente manera. En la Sección 11.1 presentamos varios conceptos básicos de la optimización multi-objetivo. En la Sección 11.2, describimos MOCell, nuestra propuesta para resolver MOPs. Nuestros resultados se presentan y se discuten en la Sección 11.3. En la Sección 11.4 se exploran algunos aspectos del diseño de MOCell para mejorar su comportamiento, y se compara la mejor de las versiones propuestas de MOCell con dos algoritmos del estado del arte en optimización multi-objetivo. Finalmente, en la Sección 11.5 mostramos nuestras principales conclusiones.

11.1. Fundamentos de la Optimización Multi-objetivo

En esta sección incluimos unos conceptos básicos de la optimización multi-objetivo para familiarizar al lector con este campo. En concreto, definimos los conceptos MOP, optimalidad de Pareto, dominancia de Pareto, conjunto óptimo de Pareto y el frente de Pareto. En estas definiciones asumimos, sin pérdida de generalidad, la minimización de todos los objetivos. Un problema general de optimización multi-objetivo (MOP) puede ser definido formalmente de la siguiente manera:

Definición 11.1 (MOP) *Encontrar un vector* $\vec{x}^* = [x_1^*, x_2^*, \ldots, x_n^*]$ *que satisfaga las* m *restricciones de desigualdad* $g_i(\vec{x}) \geq 0, i = 1, 2, \ldots, m$*, las* p *restricciones de igualdad* $h_i(\vec{x}) = 0, i = 1, 2, \ldots, p$*, y que minimice la función vector* $\vec{f}(\vec{x}) = [f_1(\vec{x}), f_2(\vec{x}), \ldots, f_k(\vec{x})]^T$*, donde* $\vec{x} = [x_1, x_2, \ldots, x_n]^T$ *es el vector de decisión de variables.*

El conjunto de todos los valores que satisfacen las restricciones define la *región de soluciones factibles* Ω y cualquier punto en $\vec{x} \in \Omega$ es una *solución factible.*

Como se mencionó anteriormente, buscamos los *óptimos de Pareto.* Su definición formal se da a continuación:

Definición 11.2 (Optimalidad de Pareto) *Un punto* $\vec{x}^* \in \Omega$ *es un óptimo de Pareto si para cada* $\vec{x} \in \Omega$ *y* $I = \{1, 2, \ldots, k\}$*, o bien* $\forall_{i \in I} (f_i(\vec{x}) = f_i(\vec{x}^*))$ *o bien hay al menos un* $i \in I \mid f_i(\vec{x}) > f_i(\vec{x}^*)$*.*

Esta definición dice que $\vec{x}^*$ es un óptimo de Pareto si no existe ningún vector factible $\vec{x}$ que mejore algún criterio sin causar simultáneamente un empeoramiento en al menos otro criterio. Otra importante definición asociada a la optimalidad de Pareto es la siguiente:

Definición 11.3 (Dominancia de Pareto) *Un vector* $\vec{u} = (u_1, \ldots, u_k)$ *se dice que domina a otro vector* $\vec{v}=(v_1, \ldots, v_k)$ *(representado por* $\vec{u} \preccurlyeq \vec{v}$*) si y sólo si* $\vec{u}$ *es parcialmente menor que* $\vec{v}$*, es decir,* $\forall i \in \{1, \ldots, k\},\ u_i \leq v_i \ \wedge\ \exists\, i \in \{1, \ldots, k\} :\ u_i < v_i$*.*

Definición 11.4 (Conjunto Óptimo de Pareto) *Para un MOP dado* $\vec{f}(\vec{x})$*, el conjunto óptimo de Pareto se define como* $\mathcal{P}^* = \{\vec{x} \in \Omega | \neg \exists \vec{x'} \in \Omega, \vec{f}(\vec{x'}) \preccurlyeq \vec{f}(\vec{x})\}$*.*

Definición 11.5 (Frente de Pareto) *Para un MOP dado* $\vec{f}(\vec{x})$ *y su conjunto óptimo de Pareto* $\mathcal{P}^*$*, el frente de Pareto se define como* $\mathcal{PF}^* = \{\vec{f}(\vec{x}), \vec{x} \in \mathcal{P}^*\}$*.*

Obtener el frente de Pareto de un MOP es la meta principal de la optimización multi-objetivo. De todas maneras, dado que un frente de Pareto puede contener un gran número de puntos, una buena solución debe contener un número limitado de ellos, que se encuentren lo más cerca posible del frente de Pareto exacto, y que se estén uniformemente esparcidos, para que sean de la mayor utilidad posible para el experto que interprete las soluciones.

11.2. El Algoritmo MOCell

En esta sección presentamos MOCell, un algoritmo multi-objetivo basado en un modelo de cGA. Mostramos su pseudocódigo en el Algoritmo 11.1. Podemos observar que este algoritmo es muy parecido al del cGA canónico (mostrado en el Algoritmo 2.2). Una de las principales diferencias entre los dos algoritmos es la existencia de un *frente de Pareto* (Definición 11.5) en el caso del multi-objetivo. El frente de Pareto es sólo una población adicional (el archivo externo) compuesto por un número de soluciones encontradas no dominadas, ya que tiene un tamaño máximo. Para poder manejar la inserción de soluciones en el frente de Pareto con el objeto de obtener un conjunto diverso hemos utilizado un estimador de densidad basado en la distancia de *crowding* (propuesto en NSGA-II [80]). Este método también se usa para eliminar las soluciones del archivo cuando se llena.

MOCell empieza creando un frente de Pareto vacío (línea 2 del Algoritmo 11.1). Los individuos se sitúan en una rejilla toroidal de 2 dimensiones, y se les va aplicando sucesivamente el ciclo reproductor (líneas 4 a 12) hasta que alcanzamos la condición de parada (línea 3). Así, para cada individuo, el algoritmo consiste en seleccionar dos padres de su vecindario, recombinarlos para obtener un descendiente, mutarlo, evaluar el individuo resultante e insertarlo tanto en la población auxiliar (si no está dominado por el individuo actual) como en el frente de Pareto. Finalmente, después de cada generación, reemplazamos la población antigua por la auxiliar (línea 13), y se invoca un proceso de realimentación para reemplazar un número fijo de individuos de la población escogidos aleatoriamente por soluciones del archivo (línea 14). Hemos incorporado en MOCell un mecanismo que trata con restricciones para afrontar problemas que las posean. El mecanismo es el mismo que utiliza NSGA-II, según el cual el individuo que viole un menor número de soluciones se considera mejor, y a igualdad de restricciones insatisfechas los individuos se comparan en función del valor de fitness.

Algoritmo 11.1 Pseudocódigo de MOCell.

```
1. proc Evoluciona(mocell)          //Parámetros del algoritmo en 'mocell'
2. Frente_pareto = Crear_Frente() //Crea un frente de Pareto vacío
3. mientras !CondiciónParada() hacer
4.   para individuo ← 1 hasta mocell.Tamaño_Población hacer
5.     vecinos ← ObtenerVecindario(mocell,posición(individuo));
6.     padres ← Selección(vecinos);
7.     descendiente ← Recombinación(mocell.Pc,padres);
8.     descendiente ← Mutación(mocell.Pm,descendiente);
9.     Evaluar(descendiente);
10.    Insertar(posición(individuo),descendiente,mocell,poblac_auxiliar);
11.    InsertarFrentePareto(individuo, Frente_pareto);
12.  fin para
13.  mocell.poblac ← poblac_auxiliar;
14.  mocell.poblac ← Retroalimentar(mocell,Frente_Pareto);
15. fin mientras
16. fin proc Evoluciona;
```

11.3. Resultados Computacionales

Esta sección está dedicada a la evaluación de MOCell. Por ello, hemos escogido varios problemas de prueba de la literatura especializada, y para mostrar lo competitivo que es, hemos decidido compararlo con los dos algoritmos más representativos del estado del arte, NSGA-II y SPEA2. Después, comentaremos brevemente las principales características de esos algoritmos, incluyendo los parámetros utilizados en los experimentos.

El algoritmo NSGA-II se propuso por Deb et al. en [80]. Se caracteriza por una ordenación de los individuos de Pareto y por la utilización de la distancia de *crowding* como estimador de densidad. Hemos utilizado en nuestras comparaciones la implementación del NSGA-II original de Deb[1]. Específicamente, utilizamos la versión de codificación real del algoritmo y los parámetros propuestos en [80]. Las probabilidades de recombinación y mutación son $p_c = 0.9$ y $p_m = 1/n$ (donde n es el número de variables de decisión), respectivamente. Los operadores de recombinación y mutación son la recombinación binaria simulada –SBX– y mutación polinómica [79], con índices de distribución de $\eta_c = 20$ y $\eta_m = 20$, respectivamente. El tamaño de la población y del archivo es de 100 individuos. El algoritmo se detiene tras realizar 25000 evaluaciones de la función objetivo.

SPEA2 fue propuesto por Zitler et al. en [301]. En este algoritmo, cada individuo tiene asignado un valor de fitness que es la suma de su valor de fitness original y una estimación de densidad basada en la distancia al k-ésimo vecino más cercano. Como en el caso de NSGA-II, hemos utilizado la implementación de los autores de SPEA2[2]. El algoritmo se implementa con la biblioteca PISA [45]. De todas maneras, la implementación de SPEA2 no contempla el uso de problemas con restricciones, así que nos vimos forzados a modificar la implementación original para incluir el mismo mecanismo de restricción utilizado en NSGA-II y MOCell. Hemos utilizado los siguientes valores para los parámetros. Tanto la población como el archivo tienen un tamaño de 100 individuos, y los operadores de recombinación y mutación

[1] NSGA-II está disponible para descargarse en: `http://www.iitk.ac.in/kangal/soft.htm`

[2] SPEA2 está disponible en: `http://www.tik.ee.ethz.ch/pisa/selectors/spea2/spea2.html`

Tabla 11.1: Parametrización utilizada en MOCell.

Tamaño de la Población	100 Individuos (10×10)
Máximo Número de Evaluaciones	25000
Condición de Parada	25000 Evaluaciones de la Función Objetivo
Vecindario	C9
Selección de los Padres	Torneo Binario + Torneo Binario
Recombinación	SBX, $p_c = 1.0$
Mutación	Polinomial, $p_m = 1.0/L$ (L = Longitud del Individuo)
Reemplazo	Reempl_si_Mejor (NSGA-II Crowding)
Tamaño de Archivo	100 Individuos
Estimador de Densidad	Distancia de Crowding
Realimentación	20 Individuos

son los mismos que usa NSGA-II, utilizando los mismos valores que conciernen a sus probabilidades de aplicación y a sus índices de distribución. Como en NSGA-II, la condición de parada es calcular 25000 evaluaciones de función.

En la Tabla 11.1 mostramos los parámetros que utiliza MOCell. Hemos escogido una rejilla toroidal cuadrada de 100 individuos para estructurar la población. El vecindario utilizado es C9, que se compone de 9 individuos: el individuo considerado y los situados a su norte, sur, este, oeste, noroeste, suroeste, noreste y sureste. También hemos utilizado SBX y mutación polinómica con los mismos índices de distribución que NSGA-II y SPEA2. Las tasas de recombinación y mutación son $p_c = 1.0$ y $p_m = 1/L$, respectivamente.

El descendiente resultante reemplaza al individuo en la posición actual únicamente si es mejor, pero, como es habitual en la optimización multi-objetivo, necesitamos definir el concepto de "mejor individuo". Nuestra aproximación es reemplazar el individuo actual si es dominado por el descendiente o si los dos son no dominados y el individuo actual tiene la peor distancia de crowding (como se define en NSGA-II) en una población compuesta por el vecindario y el descendiente. Para insertar al individuo en el frente de Pareto, las soluciones del archivo también se ordenan según la distancia de crowding; por tanto, al insertar una solución no dominada, si el frente de Pareto ya está completo, la solución con el peor valor de la distancia de crowding se elimina. Finalmente, después de cada iteración, 20 individuos escogidos al azar en la población se reemplazan por las 20 mejores soluciones del archivo externo de acuerdo con la distancia de crowding (mecanismo de realimentación).

Para nuestras pruebas hemos seleccionado problemas (tanto con restricciones como sin ellas) que han sido utilizados en la mayoría de estudios de este área: Schaffer, Fonseca, Kursawe, ZDT1, ZDT2, ZDT3, ZDT4, ZDT6 (sin restricciones), Osyczka2, Tanaka, ConstrEx y Srinivas (estos 4 últimos tienen restricciones). Todos estos problemas se describen en el Apéndice A. En la Sección 11.3.1 pasamos a comentar las métricas que hemos utilizado para comparar los algoritmos, mientras que los resultados obtenidos se muestran en la Sección 11.3.2.

11.3.1. Métricas de Rendimiento

Para mostrar el rendimiento de los algoritmos en los problemas de prueba, se tienen normalmente en cuenta dos aspectos distintos: (i) minimizar la distancia del frente de Pareto que el algoritmo propuesto genera con respecto al frente de Pareto exacto y (ii) maximizar la dispersión de las soluciones encontradas, para que podamos tener una distribución de vectores tan suave y uniforme como sea posible. Para determinar el primer aspecto normalmente se necesita saber la posición exacta del verdadero frente de Pareto; hemos obtenido estos frentes de [215], donde los autores utilizan una estrategia de búsqueda enumerativa para calcular dichos frentes, que se encuentran disponibles en `http://neo.lcc.uma.es/software/esam`. En el caso de la familia de problemas ZDTx, los frentes pueden ser fácilmente calculados porque se conocen sus soluciones.

Para la comparación de los resultados presentados en esta sección hemos propuesto el estudio de dos métricas distintas. Una de ellas, la *distancia generacional* (GD) cuantifica la distancia del frente de Pareto obtenido al frente óptimo al problema; mientras que la otra métrica seleccionada, *dispersión* (Δ) es una medida de la diversidad de las soluciones del frente. Para más detalles acerca de estas métricas, consulte el Capítulo 12.

11.3.2. Discusión de los Resultados

Los resultados están resumidos en las tablas 11.2 (GD) y 11.3 (Δ), y resaltamos el mejor resultado para cada problema en **negrita**. Para cada problema, realizamos 100 ejecuciones independientes. Las tablas incluyen la media, $\bar{x}$, y la desviación estándar, σ_n, de los resultados. Como se ha venido haciendo a lo largo de todo este trabajo de tesis, hemos realizado un análisis estadístico de los resultados ya que tratamos con algoritmos estocásticos. Consiste en los siguientes pasos. Primero se realiza un análisis de Kolmogorov-Smirnov para comprobar si los valores de los resultados siguen una distribución normal o no. Si la siguen, se le realiza un análisis de ANOVA, si no, le aplicaríamos un análisis de Kruskal-Wallis. En este trabajo siempre hemos considerado un nivel de confidencia del 95 % en las pruebas estadísticas. El símbolo '+' significa en la Tablas 11.2 y 11.3 que la diferencia entre los valores de los tres algoritmos para un problema dado tiene confianza estadística (p-valor por debajo de 0.05).

Tabla 11.2: Media ($\bar{x}$) y desviación estándar (σ_n) de la medida de la convergencia GD.

Problema	**MOCell** $\bar{x}_{\pm\sigma_n}$	**NSGA-II** $\bar{x}_{\pm\sigma_n}$	**SPEA2** $\bar{x}_{\pm\sigma_n}$	**Test**
Schaffer	2.408e-4$_{\pm 1.79e-5}$	**2.328e-4$_{\pm 1.19e-5}$**	2.365e-4$_{\pm 1.06e-5}$	+
Fonseca	**1.983e-4$_{\pm 1.60e-5}$**	4.683e-4$_{\pm 3.95e-5}$	2.251e-4$_{\pm 2.37e-5}$	+
Kursawe	**1.435e-4$_{\pm 1.01e-5}$**	2.073e-4$_{\pm 2.22e-5}$	1.623e-4$_{\pm 1.51e-5}$	+
Zdt1	4.057e-4$_{\pm 6.57e-5}$	2.168e-4$_{\pm 3.57e-5}$	**1.992e-4$_{\pm 1.34e-5}$**	+
Zdt2	2.432e-4$_{\pm 9.29e-5}$	1.714e-4$_{\pm 3.80e-5}$	**1.095e-4$_{\pm 5.37e-5}$**	+
Zdt3	2.540e-4$_{\pm 2.78e-5}$	**2.199e-4$_{\pm 3.72e-5}$**	2.336e-4$_{\pm 1.34e-5}$	+
Zdt4	8.273e-4$_{\pm 1.85e-3}$	**4.888e-4$_{\pm 2.59e-4}$**	6.203e-2$_{\pm 3.94e-2}$	+
Zdt6	2.106e-3$_{\pm 3.33e-4}$	1.001e-3$_{\pm 8.66e-5}$	**8.252e-4$_{\pm 5.15e-5}$**	+
ConstrEx	**1.968e-4$_{\pm 2.29e-5}$**	2.903e-4$_{\pm 3.19e-5}$	2.069e-4$_{\pm 1.78e-5}$	+
Srinivas	**5.147e-5$_{\pm 1.51e-5}$**	1.892e-4$_{\pm 3.01e-5}$	1.139e-4$_{\pm 1.98e-5}$	+
Osyczka2	2.678e-3$_{\pm 5.30e-3}$	**1.071e-3$_{\pm 1.33e-4}$**	6.149e-3$_{\pm 1.14e-2}$	+
Tanaka	7.494e-4$_{\pm 7.09e-5}$	1.214e-3$_{\pm 7.95e-5}$	**7.163e-4$_{\pm 7.13e-5}$**	+

Tabla 11.3: Media ($\bar{x}$) y desviación estándar (σ_n) de la medida de la diversidad Δ.

Problema	**MOCell** $\bar{\mathbf{x}}_{\pm\sigma_\mathbf{n}}$	**NSGA-II** $\bar{\mathbf{x}}_{\pm\sigma_\mathbf{n}}$	**SPEA2** $\bar{\mathbf{x}}_{\pm\sigma_\mathbf{n}}$	**Test**
Schaffer	2.473e-1$_{\pm 3.11e-2}$	4.448e-1$_{\pm 3.62e-2}$	**1.469e-1$_{\pm 1.14e-2}$**	+
Fonseca	**9.695e-2$_{\pm 1.08e-2}$**	3.596e-1$_{\pm 2.83e-2}$	1.445e-1$_{\pm 1.28e-2}$	+
Kursawe	**4.121e-1$_{\pm 4.32e-3}$**	5.460e-1$_{\pm 2.41e-2}$	4.390e-1$_{\pm 8.94e-3}$	+
Zdt1	**1.152e-1$_{\pm 1.40e-2}$**	3.645e-1$_{\pm 2.91e-2}$	1.684e-1$_{\pm 1.29e-2}$	+
Zdt2	**1.120e-1$_{\pm 1.61e-2}$**	3.644e-1$_{\pm 3.03e-2}$	1.403e-1$_{\pm 6.71e-2}$	+
Zdt3	**6.998e-1$_{\pm 3.25e-2}$**	7.416e-1$_{\pm 2.25e-2}$	7.040e-1$_{\pm 1.78e-2}$	+
Zdt4	1.581e-1$_{\pm 6.14e-2}$	3.651e-1$_{\pm 3.32e-2}$	**1.049e-1$_{\pm 1.71e-1}$**	+
Zdt6	1.859e-1$_{\pm 2.33e-2}$	2.988e-1$_{\pm 2.48e-2}$	**1.728e-1$_{\pm 1.16e-2}$**	+
ConstrEx	**1.323e-1$_{\pm 1.32e-2}$**	4.212e-1$_{\pm 3.52e-2}$	5.204e-1$_{\pm 1.58e-2}$	+
Srinivas	**6.191e-2$_{\pm 8.63e-3}$**	3.680e-1$_{\pm 3.02e-2}$	1.628e-1$_{\pm 1.25e-2}$	+
Osyczka2	**2.237e-1$_{\pm 3.50e-2}$**	4.603e-1$_{\pm 5.58e-2}$	3.145e-1$_{\pm 1.35e-1}$	+
Tanaka	**6.629e-1$_{\pm 2.76e-2}$**	7.154e-1$_{\pm 2.35e-2}$	6.655e-1$_{\pm 2.74e-2}$	+

Primero consideremos la métrica *GD* (Tabla 11.2). Podemos observar que los tres algoritmos comparados obtienen el mejor resultado en 4 de los 12 problemas estudiados. De acuerdo con estos resultados no podemos decidir un algoritmo ganador considerando la convergencia, aunque nos permiten concluir que MOCell es un algoritmo competitivo comparado con NSGA-II y SPEA2. De hecho, si consideramos sólo los problemas con restricciones estudiados, el lector puede observar que MOCell se comporta mejor que los otros algoritmos comparados, ya que muestra los mejores resultados para ConstrEx y Srinivas, mientras que es la segunda mejor aproximación para los otros dos problemas.

Con referencia a la medida de la dispersión (Tabla 11.3), los resultados indican que MOCell claramente mejora a los otros dos algoritmos con respecto a la diversidad de los frentes de Pareto obtenidos, ya que consigue los mejores valores en 9 de los 12 problemas. Además, MOCell presenta los mejores resultados para todos los problemas con restricciones estudiados. Destaca que NSGA-II no puede alcanzar el mejor valor para la medida de la dispersión en ningún problema.

Para mostrar gráficamente nuestros resultados, dibujamos en la Figura 11.1 tres frentes obtenidos por MOCell, NSGA-II, y SPEA2 para el problema ConstrEx, junto con el frente óptimo de Pareto obtenido por el algoritmo enumerativo. Los frentes seleccionados son aquellos que tienen la mejor diversidad (menor valor de Δ) entre los 100 producidos por cada técnica para ese problema. Podemos observar que el conjunto de soluciones no dominadas que MOCell alcanza una dispersión y convergencia casi perfectas. Note que SPEA2 obtiene una diversidad muy buena para los valores de $f_1(\vec{x})$ menores que 0.66 (parecido a la solución de MOCell), pero sólo encuentra 10 soluciones cuando $f_1(\vec{x}) \in [0.66, 1.0]$. Con referencia a NSGA-II, su frente no tiene ese problema, pero a simple vista se nota que tiene peor diversidad que MOCell.

También queremos señalar que, en lo concerniente a la diversidad, MOCell no sólo es el mejor de los tres algoritmos analizados, sino que las diferencias entre los valores de la dispersión son en general notables en comparación con el resto de los algoritmos.

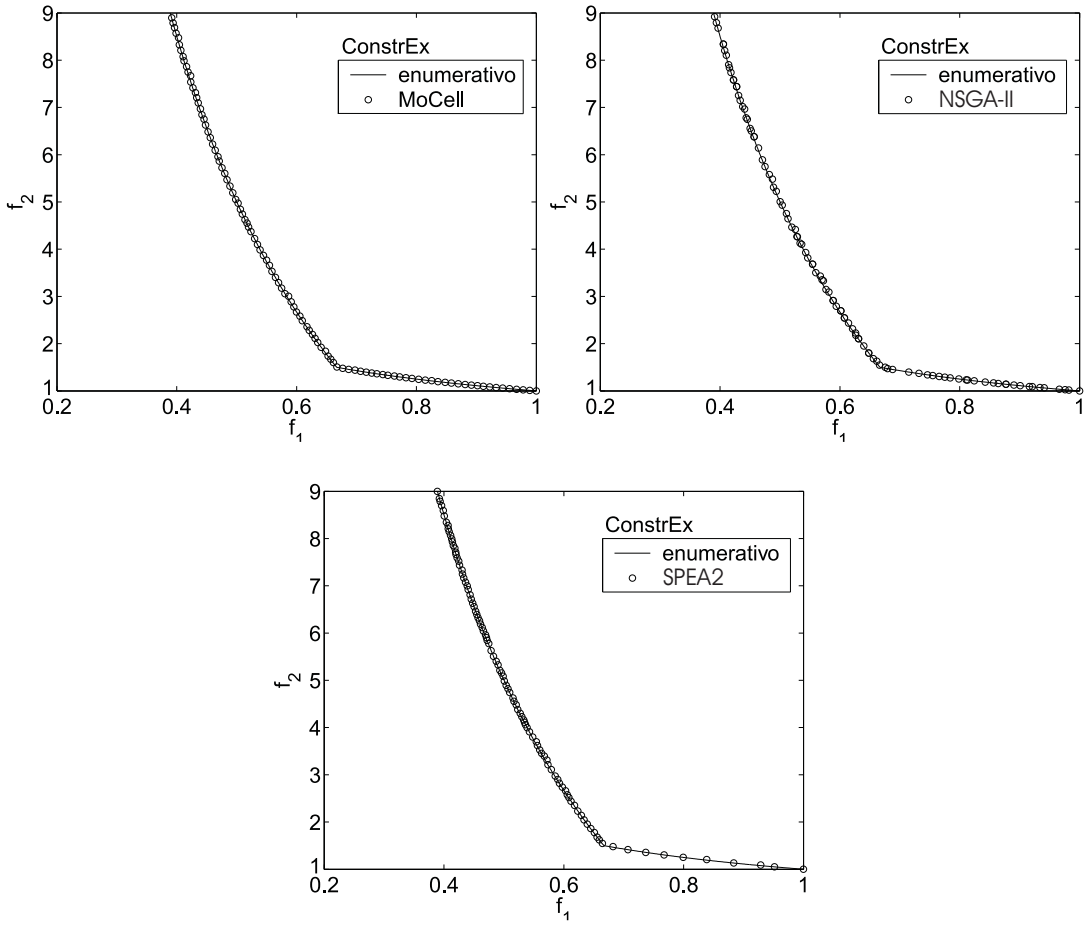


Figura 11.1: MOCell encuentra una convergencia y diversidad de soluciones mejor que NSGA-II y SPEA2 para el problema ConstrEx.

11.4. Nuevos Aspectos de Diseño en MOCell

Como vimos en la Sección 11.3.2, MOCell es un algoritmo realmente competitivo con los principales algoritmos del estado del arte en optimización multi-objetivo. Aun así, consideramos que el algoritmo es susceptible de mejora, por lo que en esta sección se presentan algunos aspectos de diseño que aplicamos a MOCell con el objetivo de mejorar su comportamiento.

Los aspectos de diseño de MOCell a los que prestamos atención en este trabajo son la sincronía en la política de actualización de los individuos, la forma de llevar a cabo la realimentación de individuos y las estrategias de reemplazo de los individuos en la población. En total, presentamos 5 nuevas variantes algorítmicas a la propuesta original de MOCell. Alguna de las nuevas estrategias propuestas nos conducen a versiones de cGAs no ortodoxas, pero nuestro objetivo aquí es encontrar un algoritmo que mejore la versión actual de MOCell, para así presentar a la comunidad científica un nuevo algoritmo altamente competitivo en el estado del arte de la optimización multi-objetivo. A continuación proponemos las novedades que aplicamos en MOCell en este estudio:

- **Sincronía.** El algoritmo MOCell estudiado en la Sección 11.3.2 actualiza todos los individuos en paralelo (estrategia de actualización síncrona). En esta sección nos planteamos el estudio de la influencia que tiene en MOCell el uso de una estrategia asíncrona para actualizar la población. De entre las estrategias asíncronas existentes (presentadas en el Capítulo 2), hemos seleccionado la más simple, LS, que consiste en ir actualizando los individuos en el mismo orden en que aparecen en la población, línea a línea.
- **Realimentación.** En MOCell, un número determinado de individuos del archivo reemplaza a un conjunto de individuos aleatorios de la población al final de cada generación. Este modelo tiene varias limitaciones, puesto que en la realimentación se pueden perder individuos importantes de la población (tanto buenos individuos como individuos que proporcionan diversidad). Además, existe un riesgo elevado de eliminar todos (o gran parte de) los individuos de la población tras unas pocas generaciones, especialmente si se realimenta un número elevado de individuos. Por tanto, proponemos un nuevo modelo de realimentación implícito alternativo en MOCell que se lleva a cabo en la etapa de selección. Para ello, forzamos a que uno de los padres sea seleccionado del archivo, mientras que el otro se selecciona del vecindario del individuo actual.
- **Reemplazo.** El modelo de cGA canónico establece que el descendiente reemplaza, con una política dada, al individuo actual. Nosotros proponemos en este trabajo considerar para el reemplazo, en el contexto de un cGA asíncrono, el vecindario entero en lugar del individuo actual únicamente, por lo que el peor individuo del vecindario es el que será considerado para el reemplazo siguiendo la política considerada.

Teniendo en cuenta los aspectos de diseño mencionados anteriormente, proponemos cinco nuevas configuraciones de nuestro algoritmo. Estas propuestas se describen a continuación:

- **sMOCell1**: Es la versión de MOCell propuesta en la Sección 11.2, y estudiada en la Sección 11.3.2.
- **sMOCell2**: MOCell sustituyendo la retroalimentación al final de cada generación por la realimentación implícita en la selección de padres.
- **aMOCell1**: Versión asíncrona de sMOCell1.
- **aMOCell2**: aMOCell1 + realimentación en la selección de padres.
- **aMOCell3**: aMOCell1 + reemplazo del peor vecino.
- **aMOCell4**: Combinación de aMOCell2 y aMOCell3.

Se procede ahora a comparar las seis versiones de MOCell propuestas. Para ello, utilizamos las mismas métricas estudiadas en la Sección 11.3.2, GD (como métrica de convergencia) y Δ (como medida de diversidad), más la métrica *hipervolumen* –HV– (consulte el Capítulo 4 para más detalles acerca de las métricas). Con el fin de hacer más completo nuestro estudio, hemos seleccionado para nuestras comparaciones un nuevo (y más reciente) conjunto de problemas, llamado WFG (descrito en el Apéndice A), además de las funciones ZDT1, ZDT2, ZDT3, ZDT4 y ZDT6, ya estudiadas en la Sección 11.3.2.

Mostramos en las tablas 11.4 a 11.6 los resultados de la comparación de las seis versiones de MOCell para las tres métricas propuestas. Se muestran la mediana ($\tilde{x}$) de los resultados tras 100 ejecuciones y el rango intercuartílico (IQR). El mejor resultado de todos los algoritmos para cada métrica en cada problema se resalta en **negrita**. Hemos aplicado los mismos tests estadísticos que en la Sección 11.3.2.

En la Tabla 11.4 mostramos la comparación de los seis algoritmos en términos de la métrica GD. En este caso, aMOCell4 obtiene los mejores (menores) valores en seis de los catorce problemas estudiados; además, existe confianza estadística en cinco de ellos (como se puede observar prestando atención a los símbolos "+" en la última columna). Si comparamos las versiones síncronas con las asíncronas, éstas últimas calculan conjuntos de soluciones no dominadas más próximos a los frentes de Pareto exactos de los MOPs. De hecho, las versiones asíncronas de MOCell alcanzan el mejor valor de GD en 11 de los 14 problemas.

Los resultados de la métrica Δ se muestran en la Tabla 11.5. Una vez más, aMOCell4 consigue los mejores valores en 6 (de los 14) MOPs. En términos de esta métrica, las versiones asíncronas de MOCell también mejoran a las síncronas, alcanzando la mejor distribución de soluciones no dominadas en el frente de Pareto en 13 de los 14 problemas.

Finalmente, concluimos que la métrica HV refuerza los resultados de las dos métricas anteriores (Tabla 11.6). En primer lugar, aMOCell4 alcanza los mejores (mayores) valores para 8 de los 14 MOPS y, en segundo lugar, las versiones asíncronas derrotan a las síncronas en todos los problemas, con la excepción del problema WFG6.

Si prestamos atención a la política de reemplazo utilizada, los resultados muestran que seleccionar uno de los padres del archivo permite obtener mejores resultados que la política original de realimentación de MOCell (en la que un número determinado de soluciones son copiadas en la población) tanto en el caso síncrono (sMOCell2 mejora a sMOCell1 en 12, 10, y 9 problemas para las métricas GD, Δ, y HV, respectivamente) como en el asíncrono (aMOCell2 es mejor que aMOCell1 en 11, 10, y 11 problemas para las métricas GD, Δ, y HV, respectivamente).

También queremos destacar que, aunque las diferencias en los valores entre las distintas versiones de MOCell son muy pequeñas para todas las métricas, esto se debe al proceso de normalización que se aplica a los conjuntos de soluciones no dominadas antes de calcular las métricas. De hecho, las diferencias en los valores de las comparaciones mostradas son significativas con confianza estadística (como muestran los símbolos "+" en la última columna de las tablas). Consecuentemente, podemos afirmar que la combinación de las estrategias de reemplazo y retroalimentación erigen a aMOCell4 como la mejor de las seis configuraciones de algoritmos comparados para los problemas estudiados. A continuación, con el fin de determinar cómo de competitiva es esta nueva versión de MOCell, procedemos a compararlo con NSGA-II y SPEA2. En este caso hemos utilizado las versiones desarrolladas en jMetal [92] de estos algoritmos, de forma que todos los algoritmos comparados comparten las mismas implementaciones de los métodos y operadores empleados. Esto nos permite realizar comparaciones más justas que las realizadas con los algoritmos originales en la Sección 11.3.2, y demostramos así que las diferencias obtenidas en los resultados son debidas a los distintos modelos algorítmicos utilizados. Además, la competitividad de MOCell con las versiones originales de NSGA-II y SPEA2 quedó ya demostrada en la Sección 11.3.2.

Se muestran en las tablas 11.7, 11.8, y 11.9 los resultados de aMOCell4, NSGA-II y SPEA2 para las métricas GD, Δ, y HV, respectivamente. Si comenzamos analizando la proximidad de los frentes encontrados al frente de Pareto, la Tabla 11.7 muestra que nuestra aproximación celular obtiene los mejores (menores) valores para la métrica GD en 10 de los 14 MOPs estudiados. aMOCell4 resulta especialmente apropiado para la familia de problemas ZDT, para los que es el mejor algoritmo en los cinco problemas. En el caso de las funciones WFG, NSGA-II y SPEA2 obtienen cada uno de ellos los frentes más cercanos al exacto en 2 problemas (de 9), mientras que aMOCell4 es capaz de encontrar los mejores resultados para los 5 problemas restantes. Por tanto, es claro que, en términos de la convergencia, el algoritmo celular mejora a NSGA-II y SPEA2 en el conjunto de problemas considerado. Nótese que, además, estas afirmaciones están sustentadas con la confianza estadística de los resultados (símbolos "+" en la última columna).

Tabla 11.4: Comparación de las diferentes versiones de MOCell: media y rango intercuartílico de la métrica *GD*.

MOP	sMOCell1 $\tilde{x}_{IQR}$	sMOCell2 $\tilde{x}_{IQR}$	aMOCell1 $\tilde{x}_{IQR}$	aMOCell2 $\tilde{x}_{IQR}$	aMOCell3 $\tilde{x}_{IQR}$	aMOCell4 $\tilde{x}_{IQR}$	
ZDT1	6.288e-4 $_{1.5e-4}$	2.749e-4 $_{7.5e-5}$	4.207e-4 $_{7.2e-5}$	2.518e-4 $_{4.5e-5}$	2.222e-4 $_{4.1e-5}$	**1.753e-4 $_{2.0e-5}$**	+
ZDT2	5.651e-4 $_{2.0e-4}$	1.778e-4 $_{1.1e-4}$	2.884e-4 $_{1.9e-4}$	1.111e-4 $_{1.1e-4}$	1.197e-4 $_{5.3e-5}$	**5.629e-5 $_{2.5e-5}$**	+
ZDT3	3.326e-4 $_{8.5e-5}$	2.493e-4 $_{3.0e-5}$	2.644e-4 $_{4.4e-5}$	2.427e-4 $_{2.8e-5}$	2.077e-4 $_{2.0e-5}$	**2.008e-4 $_{1.8e-5}$**	+
ZDT4	9.668e-4 $_{6.4e-4}$	3.848e-4 $_{2.9e-4}$	7.847e-4 $_{5.9e-4}$	4.235e-4 $_{3.3e-4}$	6.179e-4 $_{4.0e-4}$	**3.293e-4 $_{2.0e-4}$**	+
ZDT6	3.963e-3 $_{1.3e-3}$	1.080e-3 $_{2.0e-4}$	2.397e-3 $_{7.8e-4}$	9.334e-4 $_{1.3e-4}$	8.778e-4 $_{1.3e-4}$	**6.323e-4 $_{3.4e-5}$**	+
WFG1	1.962e-4 $_{8.0e-3}$	**1.859e-4 $_{1.9e-5}$**	1.906e-4 $_{6.5e-3}$	1.889e-4 $_{2.0e-5}$	1.921e-4 $_{8.1e-3}$	2.052e-4 $_{1.0e-2}$	+
WFG2	4.408e-4 $_{1.4e-4}$	4.339e-4 $_{1.2e-4}$	4.410e-4 $_{1.3e-4}$	**4.316e-4 $_{1.3e-4}$**	4.337e-4 $_{1.3e-4}$	4.336e-4 $_{7.1e-5}$	+
WFG3	1.372e-4 $_{1.4e-5}$	1.349e-4 $_{1.5e-5}$	1.375e-4 $_{1.8e-5}$	**1.340e-4 $_{1.3e-5}$**	1.367e-4 $_{1.5e-5}$	1.354e-4 $_{1.4e-5}$	+
WFG4	6.423e-4 $_{2.2e-5}$	6.259e-4 $_{2.6e-5}$	6.396e-4 $_{2.6e-5}$	**6.252e-4 $_{2.4e-5}$**	6.341e-4 $_{2.6e-5}$	6.253e-4 $_{3.2e-5}$	+
WFG5	2.634e-3 $_{2.6e-5}$	2.633e-3 $_{1.4e-5}$	2.636e-3 $_{3.4e-5}$	**2.631e-3 $_{1.4e-5}$**	2.633e-3 $_{1.2e-5}$	2.635e-3 $_{1.1e-5}$	+
WFG6	**4.984e-4 $_{4.3e-4}$**	1.210e-3 $_{2.1e-3}$	5.146e-4 $_{7.1e-4}$	1.268e-3 $_{3.4e-3}$	5.976e-4 $_{7.1e-4}$	1.906e-3 $_{3.4e-3}$	+
WFG7	3.069e-4 $_{2.2e-5}$	3.048e-4 $_{2.3e-5}$	3.025e-4 $_{2.1e-5}$	3.038e-4 $_{2.7e-5}$	3.067e-4 $_{2.4e-5}$	**3.011e-4 $_{2.4e-5}$**	•
WFG8	1.009e-2 $_{6.6e-3}$	1.460e-2 $_{5.4e-3}$	**1.000e-2 $_{6.0e-3}$**	1.468e-2 $_{3.3e-3}$	1.434e-2 $_{5.2e-3}$	1.474e-2 $_{4.9e-3}$	+
WFG9	1.072e-3 $_{6.1e-5}$	**1.055e-3 $_{5.3e-5}$**	1.081e-3 $_{5.5e-5}$	1.067e-3 $_{6.6e-5}$	1.067e-3 $_{5.8e-5}$	1.065e-3 $_{6.0e-5}$	+

Tabla 11.5: Comparación de las diferentes versiones de MOCell: media y rango intercuartílico de la métrica Δ.

MOP	sMOCell1 $\tilde{x}_{IQR}$	sMOCell2 $\tilde{x}_{IQR}$	aMOCell1 $\tilde{x}_{IQR}$	aMOCell2 $\tilde{x}_{IQR}$	aMOCell3 $\tilde{x}_{IQR}$	aMOCell4 $\tilde{x}_{IQR}$	
ZDT1	1.541e-1 $_{2.1e-2}$	9.645e-2 $_{1.4e-2}$	1.345e-1 $_{1.9e-2}$	9.161e-2 $_{1.3e-2}$	1.011e-1 $_{1.7e-2}$	**7.493e-2 $_{1.3e-2}$**	+
ZDT2	1.753e-1 $_{3.8e-2}$	9.907e-2 $_{1.9e-2}$	1.363e-1 $_{3.5e-2}$	9.089e-2 $_{2.4e-2}$	1.003e-1 $_{2.1e-2}$	**8.095e-2 $_{1.3e-2}$**	+
ZDT3	7.106e-1 $_{7.5e-3}$	7.073e-1 $_{7.4e-3}$	7.091e-1 $_{7.8e-3}$	7.069e-1 $_{7.0e-3}$	**7.039e-1 $_{4.0e-3}$**	7.054e-1 $_{5.4e-3}$	+
ZDT4	1.964e-1 $_{9.1e-2}$	1.257e-1 $_{3.6e-2}$	1.854e-1 $_{6.3e-2}$	1.324e-1 $_{4.5e-2}$	1.419e-1 $_{3.1e-2}$	**1.089e-1 $_{2.5e-2}$**	+
ZDT6	3.806e-1 $_{1.1e-1}$	1.513e-1 $_{2.5e-2}$	2.953e-1 $_{7.3e-2}$	1.363e-1 $_{1.9e-2}$	1.536e-1 $_{1.8e-2}$	**9.234e-2 $_{1.1e-2}$**	+
WFG1	5.469e-1 $_{9.3e-2}$	5.653e-1 $_{7.6e-2}$	5.298e-1 $_{1.0e-1}$	5.571e-1 $_{7.3e-2}$	**4.679e-1 $_{1.2e-1}$**	5.790e-1 $_{8.6e-2}$	+
WFG2	7.490e-1 $_{1.1e-2}$	**7.468e-1 $_{1.0e-2}$**	7.474e-1 $_{1.1e-2}$	7.468e-1 $_{9.9e-3}$	7.468e-1 $_{1.0e-2}$	7.471e-1 $_{8.5e-3}$	+
WFG3	3.698e-1 $_{9.8e-3}$	3.657e-1 $_{8.0e-3}$	3.725e-1 $_{8.2e-3}$	**3.634e-1 $_{7.3e-3}$**	3.684e-1 $_{7.2e-3}$	3.648e-1 $_{8.7e-3}$	+
WFG4	1.349e-1 $_{1.9e-2}$	1.341e-1 $_{1.7e-2}$	1.335e-1 $_{1.7e-2}$	1.336e-1 $_{1.9e-2}$	1.335e-1 $_{1.7e-2}$	**1.333e-1 $_{1.7e-2}$**	•
WFG5	1.311e-1 $_{2.5e-2}$	1.298e-1 $_{1.7e-2}$	1.377e-1 $_{2.3e-2}$	**1.289e-1 $_{2.3e-2}$**	1.300e-1 $_{2.3e-2}$	1.293e-1 $_{1.8e-2}$	+
WFG6	1.178e-1 $_{2.1e-2}$	1.339e-1 $_{3.4e-2}$	**1.167e-1 $_{2.7e-2}$**	1.344e-1 $_{4.6e-2}$	1.190e-1 $_{2.7e-2}$	1.348e-1 $_{4.1e-2}$	+
WFG7	1.059e-1 $_{1.8e-2}$	1.096e-1 $_{1.7e-2}$	**1.033e-1 $_{1.6e-2}$**	1.069e-1 $_{2.1e-2}$	1.040e-1 $_{1.7e-2}$	1.084e-1 $_{2.1e-2}$	+
WFG8	5.596e-1 $_{6.3e-2}$	5.664e-1 $_{8.4e-2}$	5.710e-1 $_{7.2e-2}$	5.691e-1 $_{5.0e-2}$	**5.531e-1 $_{6.4e-2}$**	5.703e-1 $_{6.7e-2}$	+
WFG9	1.597e-1 $_{1.8e-2}$	1.449e-1 $_{1.8e-2}$	1.609e-1 $_{2.1e-2}$	1.482e-1 $_{1.8e-2}$	1.606e-1 $_{1.8e-2}$	**1.435e-1 $_{1.7e-2}$**	+

Tabla 11.6: Comparación de las diferentes versiones de MOCell: media y rango intercuartílico de la métrica *HV*.

MOP	sMOCell1 $\tilde{x}_{IQR}$	sMOCell2 $\tilde{x}_{IQR}$	aMOCell1 $\tilde{x}_{IQR}$	aMOCell2 $\tilde{x}_{IQR}$	aMOCell3 $\tilde{x}_{IQR}$	aMOCell4 $\tilde{x}_{IQR}$	
ZDT1	6.543e-1 $_{2.0e-3}$	6.592e-1 $_{7.3e-4}$	6.573e-1 $_{1.1e-3}$	6.595e-1 $_{7.3e-4}$	6.603e-1 $_{5.2e-4}$	**6.610e-1 $_{2.7e-4}$**	+
ZDT2	3.216e-1 $_{2.8e-3}$	3.265e-1 $_{1.6e-3}$	3.256e-1 $_{2.6e-3}$	3.274e-1 $_{1.7e-3}$	3.276e-1 $_{8.1e-4}$	**3.284e-1 $_{5.1e-4}$**	+
ZDT3	5.111e-1 $_{2.2e-3}$	5.135e-1 $_{8.2e-4}$	5.132e-1 $_{1.2e-3}$	5.137e-1 $_{8.1e-4}$	**5.152e-1 $_{2.8e-4}$**	5.152e-1 $_{4.0e-4}$	+
ZDT4	6.487e-1 $_{9.6e-3}$	6.573e-1 $_{4.3e-3}$	6.517e-1 $_{8.4e-3}$	6.568e-1 $_{4.5e-3}$	6.539e-1 $_{5.9e-3}$	**6.580e-1 $_{3.2e-3}$**	+
ZDT6	3.487e-1 $_{1.7e-2}$	3.885e-1 $_{3.1e-3}$	3.699e-1 $_{1.0e-2}$	3.909e-1 $_{2.0e-3}$	3.920e-1 $_{2.4e-3}$	**3.970e-1 $_{8.4e-4}$**	+
WFG1	5.491e-1 $_{1.1e-1}$	6.047e-1 $_{5.8e-2}$	5.906e-1 $_{1.2e-1}$	5.983e-1 $_{1.0e-1}$	**6.115e-1 $_{1.2e-1}$**	5.043e-1 $_{1.7e-1}$	+
WFG2	5.616e-1 $_{2.9e-3}$	5.616e-1 $_{2.7e-3}$	**5.616e-1 $_{2.8e-3}$**	5.616e-1 $_{2.7e-3}$	5.616e-1 $_{2.7e-3}$	5.616e-1 $_{1.1e-3}$	•
WFG3	4.420e-1 $_{2.0e-4}$	4.420e-1 $_{1.6e-4}$	4.420e-1 $_{3.0e-4}$	**4.420e-1 $_{1.6e-4}$**	4.420e-1 $_{2.5e-4}$	4.420e-1 $_{1.6e-4}$	+
WFG4	2.187e-1 $_{3.1e-4}$	2.186e-1 $_{3.2e-4}$	2.186e-1 $_{2.8e-4}$	2.186e-1 $_{2.9e-4}$	2.187e-1 $_{2.9e-4}$	**2.188e-1 $_{2.6e-4}$**	+
WFG5	1.961e-1 $_{7.5e-5}$	1.962e-1 $_{5.4e-5}$	1.961e-1 $_{7.5e-5}$	1.962e-1 $_{6.9e-5}$	1.962e-1 $_{7.4e-5}$	**1.962e-1 $_{4.7e-5}$**	+
WFG6	**2.051e-1 $_{7.0e-3}$**	1.949e-1 $_{2.8e-2}$	2.049e-1 $_{1.1e-2}$	1.940e-1 $_{4.3e-2}$	2.036e-1 $_{1.1e-2}$	1.859e-1 $_{4.2e-2}$	+
WFG7	2.104e-1 $_{1.7e-4}$	2.104e-1 $_{1.7e-4}$	2.104e-1 $_{1.6e-4}$	2.104e-1 $_{2.0e-4}$	2.104e-1 $_{2.0e-4}$	**2.105e-1 $_{1.6e-4}$**	+
WFG8	1.456e-1 $_{2.1e-2}$	1.472e-1 $_{3.0e-3}$	1.459e-1 $_{4.9e-3}$	1.466e-1 $_{2.5e-3}$	1.462e-1 $_{2.8e-3}$	**1.479e-1 $_{2.8e-3}$**	+
WFG9	2.380e-1 $_{2.2e-3}$	2.389e-1 $_{2.4e-3}$	2.375e-1 $_{3.1e-3}$	**2.390e-1 $_{2.0e-3}$**	2.380e-1 $_{2.3e-3}$	2.381e-1 $_{3.6e-3}$	+

Tabla 11.7: Comparación de aMOCell4, NSGA-II y SPEA2. Mediana y rango intercuartílico para la métrica *GD*.

MOP	**aMOCell4** $\tilde{x}_{IQR}$	**NSGA-II** $\tilde{x}_{IQR}$	**SPEA2** $\tilde{x}_{IQR}$	**Test**
ZDT1	**1.753e-4** $_{\mathbf{2.0e-5}}$	2.198e-4 $_{4.8e-5}$	2.211e-4 $_{2.8e-5}$	+
ZDT2	**5.629e-5** $_{\mathbf{2.5e-5}}$	1.674e-4 $_{4.3e-5}$	1.770e-4 $_{4.8e-5}$	+
ZDT3	**2.008e-4** $_{\mathbf{1.8e-5}}$	2.126e-4 $_{2.1e-5}$	2.320e-4 $_{2.0e-5}$	+
ZDT4	**3.293e-4** $_{\mathbf{2.0e-4}}$	4.353e-4 $_{3.2e-4}$	5.753e-4 $_{4.4e-4}$	+
ZDT6	**6.323e-4** $_{\mathbf{3.4e-5}}$	1.010e-3 $_{1.3e-4}$	1.750e-3 $_{2.9e-4}$	+
WFG1	2.052e-4 $_{1.0e-2}$	**1.967e-4** $_{\mathbf{8.3e-3}}$	6.438e-4 $_{1.0e-2}$	+
WFG2	**4.336e-4** $_{\mathbf{7.1e-5}}$	5.196e-4 $_{1.7e-4}$	4.474e-4 $_{1.2e-4}$	+
WFG3	**1.354e-4** $_{\mathbf{1.4e-5}}$	1.553e-4 $_{1.9e-5}$	1.448e-4 $_{1.2e-5}$	+
WFG4	**6.253e-4** $_{\mathbf{3.2e-5}}$	6.870e-4 $_{1.4e-4}$	6.377e-4 $_{3.0e-5}$	+
WFG5	**2.635e-3** $_{\mathbf{1.1e-5}}$	2.655e-3 $_{3.2e-5}$	2.718e-3 $_{1.7e-5}$	+
WFG6	1.906e-3 $_{3.4e-3}$	5.539e-4 $_{6.5e-4}$	**4.654e-4** $_{\mathbf{6.7e-4}}$	+
WFG7	**3.011e-4** $_{\mathbf{2.4e-5}}$	3.444e-4 $_{4.7e-5}$	3.020e-4 $_{4.5e-5}$	+
WFG8	1.474e-2 $_{4.9e-3}$	**1.446e-2** $_{\mathbf{5.3e-3}}$	1.569e-2 $_{6.0e-3}$	+
WFG9	1.065e-3 $_{6.0e-5}$	1.223e-3 $_{2.1e-4}$	**9.313e-4** $_{\mathbf{9.1e-5}}$	+

Tabla 11.8: Comparación de aMOCell4, NSGA-II y SPEA2. Mediana y rango intercuartílico para la métrica Δ.

MOP	**aMOCell4** $\tilde{x}_{IQR}$	**NSGA-II** $\tilde{x}_{IQR}$	**SPEA2** $\tilde{x}_{IQR}$	**Test**
ZDT1	**7.493e-2** $_{\mathbf{1.3e-2}}$	3.753e-1 $_{4.2e-2}$	1.486e-1 $_{1.8e-2}$	+
ZDT2	**8.095e-2** $_{\mathbf{1.3e-2}}$	3.814e-1 $_{3.9e-2}$	1.558e-1 $_{2.8e-2}$	+
ZDT3	**7.054e-1** $_{\mathbf{5.4e-3}}$	7.458e-1 $_{2.0e-2}$	7.099e-1 $_{7.7e-3}$	+
ZDT4	**1.089e-1** $_{\mathbf{2.5e-2}}$	3.849e-1 $_{5.3e-2}$	2.612e-1 $_{1.7e-1}$	+
ZDT6	**9.234e-2** $_{\mathbf{1.1e-2}}$	3.591e-1 $_{4.6e-2}$	2.268e-1 $_{3.0e-2}$	+
WFG1	**5.790e-1** $_{\mathbf{8.6e-2}}$	7.170e-1 $_{4.5e-2}$	6.578e-1 $_{7.0e-2}$	+
WFG2	**7.471e-1** $_{\mathbf{8.5e-3}}$	7.968e-1 $_{1.5e-2}$	7.519e-1 $_{1.1e-2}$	+
WFG3	**3.648e-1** $_{\mathbf{8.7e-3}}$	6.101e-1 $_{3.8e-2}$	4.379e-1 $_{1.3e-2}$	+
WFG4	**1.333e-1** $_{\mathbf{1.7e-2}}$	3.835e-1 $_{4.3e-2}$	2.681e-1 $_{3.1e-2}$	+
WFG5	**1.293e-1** $_{\mathbf{1.8e-2}}$	4.077e-1 $_{4.0e-2}$	2.805e-1 $_{2.7e-2}$	+
WFG6	**1.348e-1** $_{\mathbf{4.1e-2}}$	3.807e-1 $_{4.2e-2}$	2.506e-1 $_{2.4e-2}$	+
WFG7	**1.084e-1** $_{\mathbf{2.1e-2}}$	3.836e-1 $_{4.4e-2}$	2.453e-1 $_{2.7e-2}$	+
WFG8	**5.703e-1** $_{\mathbf{6.7e-2}}$	6.472e-1 $_{5.1e-2}$	6.108e-1 $_{5.7e-2}$	+
WFG9	**1.435e-1** $_{\mathbf{1.7e-2}}$	3.994e-1 $_{3.9e-2}$	2.945e-1 $_{2.4e-2}$	+

Tabla 11.9: Comparación de aMOCell4, NSGA-II y SPEA2. Mediana y rango intercuartílico para la métrica *HV*.

MOP	**aMOCell4** $\tilde{x}_{IQR}$	**NSGA-II** $\tilde{x}_{IQR}$	**SPEA2** $\tilde{x}_{IQR}$	**Test**
ZDT1	**6.610e-1** $_{\mathbf{2.7e-4}}$	6.594e-1 $_{4.0e-4}$	6.600e-1 $_{3.5e-4}$	+
ZDT2	**3.284e-1** $_{\mathbf{5.1e-4}}$	3.261e-1 $_{4.8e-4}$	3.263e-1 $_{7.4e-4}$	+
ZDT3	**5.152e-1** $_{\mathbf{4.0e-4}}$	5.148e-1 $_{2.7e-4}$	5.141e-1 $_{3.4e-4}$	+
ZDT4	**6.580e-1** $_{\mathbf{3.2e-3}}$	6.552e-1 $_{4.7e-3}$	6.518e-1 $_{1.0e-2}$	+
ZDT6	**3.970e-1** $_{\mathbf{8.4e-4}}$	3.887e-1 $_{2.2e-3}$	3.785e-1 $_{4.3e-3}$	+
WFG1	5.043e-1 $_{1.7e-1}$	**5.140e-1** $_{\mathbf{1.5e-1}}$	4.337e-1 $_{1.4e-1}$	+
WFG2	5.616e-1 $_{1.1e-3}$	**5.631e-1** $_{\mathbf{2.9e-3}}$	5.615e-1 $_{2.9e-3}$	•
WFG3	**4.420e-1** $_{\mathbf{1.6e-4}}$	4.411e-1 $_{3.2e-4}$	4.418e-1 $_{2.2e-4}$	+
WFG4	**2.188e-1** $_{\mathbf{2.6e-4}}$	2.173e-1 $_{5.3e-4}$	2.181e-1 $_{3.4e-4}$	+
WFG5	**1.962e-1** $_{\mathbf{4.7e-5}}$	1.948e-1 $_{4.8e-4}$	1.956e-1 $_{1.5e-4}$	+
WFG6	1.859e-1 $_{4.2e-2}$	2.033e-1 $_{9.9e-3}$	**2.056e-1** $_{\mathbf{1.1e-2}}$	+
WFG7	**2.105e-1** $_{\mathbf{1.6e-4}}$	2.088e-1 $_{4.3e-4}$	2.098e-1 $_{2.7e-4}$	+
WFG8	**1.479e-1** $_{\mathbf{2.8e-3}}$	1.470e-1 $_{2.3e-3}$	1.469e-1 $_{1.7e-3}$	+
WFG9	2.381e-1 $_{3.6e-3}$	2.372e-1 $_{2.2e-3}$	**2.386e-1** $_{\mathbf{2.2e-3}}$	+

Si comparamos ahora la distribución de las soluciones no dominadas a lo largo del frente de Pareto obtenido por los algoritmos, la Tabla 11.8 muestra que aMOCell4 mejora claramente a los otros dos algoritmos comparados en todos los problemas estudiados. Las diferencias más importantes se dan en la familia de problemas ZDT, donde los valores de la métrica Δ obtenidos por aMOCell4 para los problemas ZDT1, ZDT2 y ZDT6 son un orden de magnitud menores (mejores) que los alcanzados por NSGA-II y SPEA2. Con el fin de ilustrar esto, mostramos en la Figura 11.2 el frente aproximado con el mejor valor de Δ en las 100 ejecuciones realizadas por cada uno de los algoritmos para WFG6. (Hemos seleccionado este MOP porque para él aMOCell4 no es el mejor en GD ni en HV.) Como se puede ver, aMOCell4 presenta una distribución casi perfecta de soluciones no dominadas a lo largo del frente de Pareto, mientras que existen algunos huecos en los frentes calculados por NSGA-II y SPEA2. Esto es especialmente importante cuando comparamos aMOCell4 con NSGA-II, ya que ambos están implementados en la biblioteca jMetal y, por tanto, comparten los mismos métodos de *ranking* y *crowding*. Esto muestra una mayor habilidad en la búsqueda que realiza nuestra aproximación celular.

Los resultados de la métrica HV se incluyen en la Tabla 11.9. Como en el caso de la métrica GD, aMOCell4 obtiene los mejores (mayores) valores en 10 de los 14 MOPs considerados. Además, aMOCell4 es también claramente el mejor de los algoritmos considerados para la familia ZDT y alcanza los mejores valores en 5 de los 9 problemas del conjunto WFG (NSGA-II y SPEA2 son los mejores algoritmos en dos problemas cada uno). En este punto, queremos recordar al lector una vez más que las pequeñas diferencias entre los algoritmos son debido al proceso de normalización que se le aplica a los frentes antes de calcular las métricas. De cualquier forma, estas pequeñas diferencias se corresponden con frentes de Pareto muy distintos, como se muestra en la Figura 11.2.

Si prestamos atención a los valores mostrados en las tablas 11.7 a 11.9, podemos concluir que aMOCell4 mejora claramente a NSGA-II y SPEA2 en las tres métricas estudiadas. Merece la pena resaltar los resultados obtenidos con la métrica de diversidad, para la que aMOCell4 obtiene los mejores resultados en todos los problemas estudiados. En cuanto a las otras dos métricas (tablas 11.7 y 11.9), aMOCell4 obtiene los mejores valores para 10 de los 14 MOPs. Además, podemos observar que los problemas en los que aMOCell4 no puede mejorar a los algoritmos comparados pertenecen al conjunto de problemas WFG, y especialmente mencionamos WFG6, WFG8 y WFG9, que son 3 de las 4 funciones cóncavas no separables que componen WFG, y WFG1, que es un problema convexo separable.

11.5. Conclusiones

En una primera parte de este capítulo hemos propuesto MOCell, un algoritmo genético celular para resolver problemas de optimización multi-objetivo. MOCell es una mejora de cMOGA, una versión anterior de cGA multi-objetivo desarrollada también durante la realización de esta tesis, que está presentado en [10, 21]. El algoritmo utiliza un archivo externo para almacenar los individuos no dominados encontrados durante la búsqueda. La característica más destacable de MOCell con respecto a las otras aproximaciones celulares para optimización multi-objetivo es la realimentación de los individuos desde el archivo a la población. MOCell se validó utilizando una metodología estándar que se usa actualmente en la comunidad de la optimización evolutiva multi-objetivo. El algoritmo se comparó con los dos algoritmos de optimización multi-objetivo más representativos del estado del arte, como son NSGA-II y SPEA2. Con este propósito, se escogieron doce problemas de prueba, tanto con restricciones como sin ellas, y se usaron dos métricas para asegurar el rendimiento de los algoritmos. Los resultados de las métricas revelan que MOCell es competitivo considerando la medida de la convergencia y claramente mejora a todos los propuestos con respecto a la medida de la dispersión.

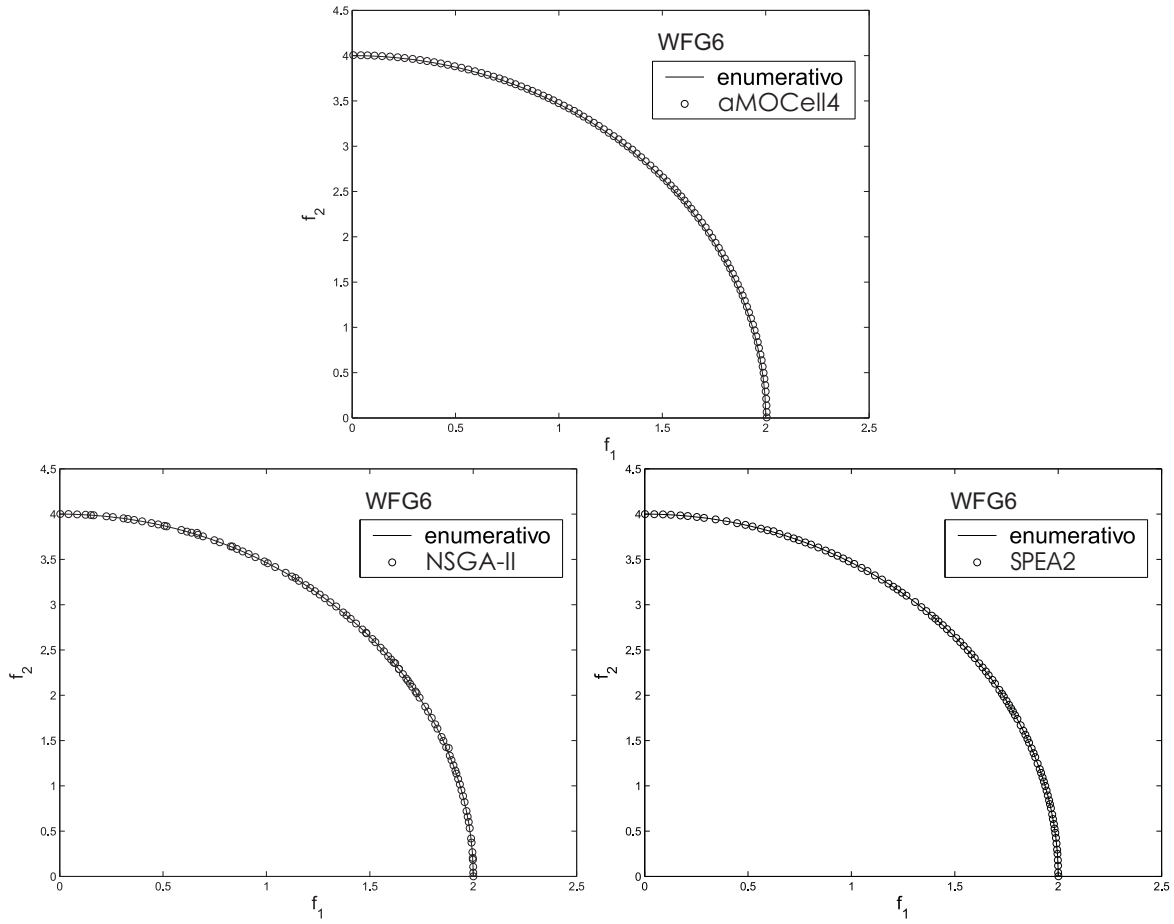


Figura 11.2: Frentes aproximados de aMOCell4, NSGA-II, y SPEA2 al resolver WFG6.

En una segunda parte del capítulo proponemos cinco variantes distintas de MOCell con el objetivo de mejorar los resultados obtenidos. En este caso, con el fin de hacer más completo nuestro estudio, hemos seleccionado un nuevo conjunto de problemas de prueba, el recientemente propuesto WFG, además de los problemas ZDT, estudiados ya en la primera parte del capítulo. Nuestras seis propuestas se han evaluado sobre tres métricas: las dos utilizadas en la primera parte más el *hipervolumen*, que tiene en cuenta a la vez tanto la diversidad como la proximidad al frente. Los resultados obtenidos indican que aMOCell4, una versión asíncrona que reemplaza al peor individuo del vecindario, y en la que se fuerza que uno de los padres sea seleccionado del archivo de soluciones no dominadas, es la mejor versión algorítmica de las seis versiones analizadas. Tras comparar esta versión con NSGA-II y SPEA2 para las tres métricas mencionadas, se ha comprobado que aMOCell4 mejora claramente a los otros dos algoritmos en el contexto de los problemas seleccionados.

Capítulo 12

Implementación de cGAs: La Biblioteca JCell

Para los trabajos que hemos llevado a cabo, se ha desarrollado una nueva biblioteca que contiene la gran mayoría de los algoritmos propuestos en los capítulos anteriores, y a la que hemos llamado JCell. Nuestra finalidad para el desarrollo de JCell es doble. Por un lado, estamos facilitando una herramienta para reproducir los resultados reportados en esta tesis con un esfuerzo realmente bajo. Por otro lado, nos proponemos proveer a la comunidad científica de una amplia biblioteca para trabajar con cEAs y los avances más recientes en este campo. Para este objetivo, consideramos que es fundamental realizar un código fácil de entender y de usar, y por tanto se ha prestado una gran atención a estos dos aspectos.

Actualmente, sólo el caso de los cGAs está implementado en JCell, pero es realmente simple e intuitivo extender JCell a otros tipos de EAs, como veremos en este capítulo. Esto permite la migración de todas las mejoras algorítmicas propuestas en esta tesis a otros campos de la computación evolutiva con un mínimo esfuerzo. Además, no sólo está disponible en JCell la opción de trabajar con poblaciones estructuradas, sino que también hay implementaciones de EAs generacionales y de estado estacionario (steady-state), lo que permite fácilmente hacer estudios de las ventajas de utilizar poblaciones estructuradas frente a poblaciones panmícticas.

En la Sección 12.1 se presenta el diseño de JCell, mientras que en la Sección 12.2 se puede encontrar una breve guía de uso del algoritmo. Por último, expresamos nuestras principales conclusiones en la Sección 12.3.

12.1. La Biblioteca JCell

En esta sección presentamos JCell. Como se ha dicho más arriba, JCell es una biblioteca genérica para trabajar con cEAs y por tanto contiene los últimos avances en el campo de los algoritmos celulares. El uso de JCell es sencillo, ya que para ello solamente es necesaria la manipulación de un simple fichero de configuración. JCell permite al usuario trabajar con problemas de optimización combinatoria, de programación con números enteros, de optimización continua, o incluso en entornos multi-objetivo, todo esto con la posibilidad de considerar o no restricciones en las soluciones al problema dado. Además, la mayoría de los modelos algorítmicos propuestos en esta tesis están también implementados en JCell.

Consideramos que JCell es una herramienta interesante y útil para investigaciones futuras, ya que permite la combinación de todas las técnicas prometedoras desarrolladas aquí. Además, su cuidado diseño, siguiendo las recomendaciones de la ingeniería del software, ofrece como resultado un código intuitivo, permitiendo al usuario hacer modificaciones y/o añadir nuevas características a la biblioteca.

Existen varios paradigmas de programación que podríamos utilizar para desarrollar JCell. La programación funcional es un paradigma de programación declarativa basado en el uso de funciones matemáticas cuyas principales características son la sencilla reutilización de funciones, el polimorfismo y el no determinismo. Otra característica interesante es que se puede demostrar formalmente la validez de un programa. La principal desventaja es su escasa eficiencia. Además, no es sencillo abordar una biblioteca de EAs como la que nos proponemos con un lenguaje de programación funcional. Sí existe sin embargo en la literatura alguna propuesta del uso de lenguajes de programación funcional para desarrollar EAs, en concreto, Yu estudia en su tesis doctoral [299] el impacto de utilizar programación funcional para el caso de la programación genética, que es la familia de EAs para la que parecen más adecuados.

El paradigma de programación imperativa es más eficiente que el funcional. Además una de sus principales características es la modularidad, y existen multitud de implementaciones de EAs en lenguajes imperativos. Sin embargo, utilizando un lenguaje orientado a objetos obtendremos un diseño mucho más sencillo y simple de nuestra biblioteca, lo que facilitará que pueda ser utilizado por otros investigadores. Además, si utilizamos un lenguaje orientado a objetos podemos beneficiarnos de características como la reutilización de código sencilla, el encapsulamiento, la herencia, la sobrecarga o el polimorfismo. Todas estas características nos permitirán desarrollar un código claro y con un diseño simple. Entre los lenguajes de programación orientados a objetos hemos seleccionado Java [2, 94] debido fundamentalmente, a su portabilidad (que es una característica muy importante teniendo en cuenta que pretendemos que JCell tenga difusión), y a la gran variedad de bibliotecas disponibles, lo que nos facilitará la tarea de programación.

Por tanto, JCell ha sido desarrollada en Java, un lenguaje de programación muy extendido y bien conocido que permite la ejecución de nuestro código en la mayoría de las plataformas y sistemas operativos sin ningún tipo de modificación. Esto es posible gracias a que Java es un lenguaje semi-interpretado, y por tanto un programa Java puede ejecutarse en cualquier tipo de computadora que tenga instalada una máquina virtual de Java (*Java Virtual Machine*), el intérprete de Java. JCell es compatible con las versiones JDK 1.5 en adelante.

En la Figura 12.1 se muestra una descripción resumida del diseño de JCell. La clase JCell es la clase principal de nuestra biblioteca, ya que contiene el método main() para ejecutar nuestro algoritmo. JCell implementa al interfaz GenerationListener, que tiene una única declaración de función (llamada generation()). Esta interfaz se utiliza para la comunicación entre la clase que lleva a cabo el ciclo reproductor del GA (EvolutionaryAlg) y JCell: tras cada generación, EvolutionaryAlg llama a la función generation() de JCell. De esta forma, el programador tiene la posibilidad de obtener la información que desee del algoritmo en cada generación, o incluso de realizar cualquier tipo de cambio durante la ejecución para obtener fácilmente variaciones sobre los GAs implementados.

JCell contiene una instancia de la clase ReadConf, que es la encargada de leer toda la parametrización requerida desde el fichero de configuración, y establece los valores correspondientes en los parámetros de EvolutionaryAlg. Por tanto, lo primero que debemos hacer cuando utilizamos la biblioteca JCell desde nuestro programa Java es una llamada a ReadConf con la ruta al fichero de configuración en el que se encuentra la parametrización deseada. La estructura de este fichero de configuración se describe en la Sección 12.2.

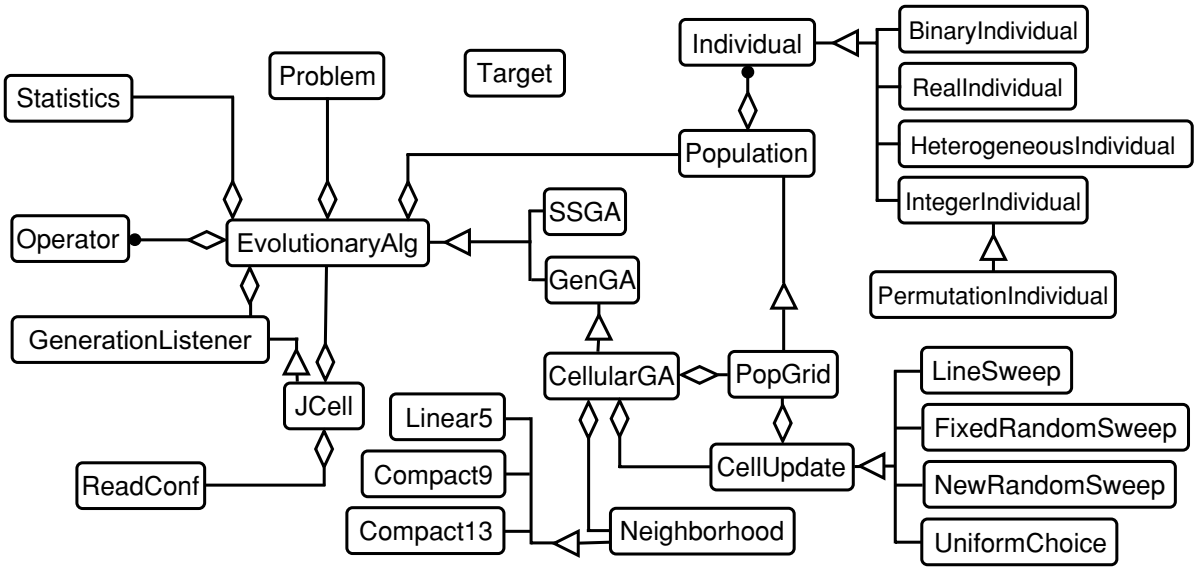


Figura 12.1: Descripción resumida del diseño de JCell.

La otra instancia de clase que contiene JCell es de EvolutionaryAlg. Como puede verse, es la clase más importante del diseño, y se trata de una clase genérica para el diseño de algoritmos evolutivos. En esta clase se almacena toda la información que necesita el algoritmo para ejecutarse, como por ejemplo la población, los operadores genéticos y sus probabilidades de aplicación, la actualización síncrona o asíncrona de los individuos, el número de generaciones (y evaluaciones) realizado, la condición de finalización del algoritmo, el modelo algorítmico a utilizar (multi-objetivo, jerárquico, o adaptativo son los que están implementados actualmente), etcétera. En los siguientes párrafos describiremos las clases de las que EvolutionaryAlg tiene referencias: Population, Neighborhood, Problem, Operator, y Statistics, además de GenerationListener, que ya fue comentada anteriormente.

Population contiene una lista lineal de objetos de tipo Individual, además de los métodos que implementan las operaciones que necesitan hacer los EAs sobre la población, como obtener o insertar un individuo, crear una población nueva, etcétera. Individual es la clase de JCell que representa a un individuo genérico de un EA, proporcionando las principales operaciones que es necesario realizar sobre individuos (obtener y establecer su valor de fitness, los valores de sus genes, etc.). Los genes del cromosoma de Individual son de tipo Object, y dependiendo del tipo de información que contendrá el genotipo del individuo ($G = \{g_1, g_2, \ldots, g_l\}$, siendo l la longitud del cromosoma) en nuestro problema, dicho individuo puede ser instanciado como:

- BinaryIndividual. Clase que hereda de Individual en la que el cromosoma está formado por valores lógicos (en Java, objetos de tipo Boolean). Este tipo de individuos se utiliza en problemas de optimización combinatoria.
- RealIndividual. Esta clase se ha creado para el caso de la optimización numérica. Los genes son números reales (del tipo Double de Java), y hereda de Individual.
- IntegerIndividual. Al igual que en las dos clases anteriores, IntegerIndividual hereda de la clase Individual. En este caso, los genes son números enteros (clase Integer).

Tabla 12.1: Tipos de individuos disponibles en JCell.

Clase	Genotipo	Hereda de
BinaryIndividual	$G = \mathbb{B}^l$	Individual
RealIndividual	$G = \mathbb{R}^l$	Individual
IntegerIndividual	$G = \mathbb{N}^l$	Individual
HeterogeneousIndividual	$G = (\mathbb{B} \cup \mathbb{R} \cup \mathbb{N})^l$	Individual
PermutationIndividual	$G = \mathbb{N}^l; g_i \neq g_j,\ 0 \leq i, j \leq l$	IntegerIndividual

- HeterogeneousIndividual. Esta clase, que al igual que las anteriores hereda de Individual, tiene la peculiaridad de que permite que un individuo tenga en su cromosoma genes de distinto tipo entre sí.
- PermutationIndividual. Es un IntegerIndividual en el que los valores de los genes son una permutación de números enteros, por lo que no se pueden dar dos genes con el mismo valor en un cromosoma.

En la Tabla 12.1 se presenta un resumen de los diferentes tipos de individuos disponibles actualmente en JCell. En concreto, se detalla, para cada tipo de individuo disponible, la clase a la que pertenece, la composición de su genotipo, y la clase de la que heredan. El valor de fitness de un individuo es una lista de objetos de tipo Double. Esta lista tiene longitud 1 cuando tratamos con problemas compuestos por una única función a optimizar, y n en el caso de optimización multi-objetivo (siendo n el número de objetivos).

Como ya se ha explicado anteriormente, un rasgo característico de los EAs celulares es que la población se encuentra estructurada mediante el establecimiento de algún tipo de relación de vecindario. La clase Neighborhood es una clase genérica utilizada para obtener los vecinos de un individuo dado, y por tanto cualquier vecindario específico que queramos definir en JCell debe heredar de esta clase. En concreto, los vecindarios Linear5, Compact9 y Compact13 han sido añadidos a JCell para el caso de trabajar con cEAs. Cabe destacar que la clase Neighborhood puede ser utilizada para definir cualquier estructura de vecindario en la población, por lo que no se debe restringir su uso únicamente al caso celular. Por otro lado, en el caso de los EAs panmícticos, como el de estado estacionario y el generacional, no existe vecindario, puesto que los padres se eligen de entre todos los individuos pertenecientes a la población (la población no se encuentra estructurada).

Todo problema que queramos resolver con JCell debe estar codificado en una clase que herede de la clase Problem. Esta clase es lo suficientemente genérica como para permitir el uso de problemas que pertenezcan a campos tan diferentes como la optimización combinatoria, continua, numérica, o incluso multi-objetivo. Además, el uso de restricciones es también posible en cualquiera de los dominios anteriores, y los valores que pueden tomar las variables del problema están restringidos dentro de un intervalo que puede ser distinto para cada uno de los genes del cromosoma. Una subclase de Problem debe contener toda la información necesaria sobre el problema considerado, como por ejemplo la función de fitness, el número de variables, que puede ser diferente a la longitud del cromosoma (ya que es posible codificar una variable con más de un gen), el número de restricciones, el mejor valor de fitness conocido para tal problema (sólo en el caso de optimización de un único objetivo), y el intervalo de los valores permitidos para cada gen.

Un objeto EvolutionaryAlg contiene un conjunto de objetos de tipo Operator. Esta clase genérica se utiliza para implementar todos los operadores de un EA, es decir, el método de selección para cada uno de los padres, los operadores de recombinación y mutación, el paso de búsqueda local (opcional), y la política de reemplazo para los individuos recientemente generados.

En cada generación, el algoritmo llama a un objeto de la clase Statistics para calcular algunas estadísticas de la ejecución en curso. El cálculo de estadísticas básicas raramente se encuentra en los códigos de otros autores, pero consideramos que es un punto esencial para el trabajo y monitorización del algoritmo. De hecho, es posible utilizar algún tipo de descriptor estadístico en algunas de las nuevas mejoras algorítmicas que hemos propuesto en esta tesis, por ejemplo, para guiar una búsqueda adaptativa (ver Capítulo 6).

Además, la clase EvolutionaryAlg contiene un método abstracto (llamado experiment()) que se encarga de aplicar el ciclo reproductor del EA. Simplemente extendiendo este método abstracto, hemos definido las clases SSGA y GenGA para implementar GAs de estado estacionario (se actualiza un individuo en cada generación) y generacional (todos los individuos son actualizados en cada generación), respectivamente. Por tanto, para extender JCell con cualquier otro tipo de EA, simplemente tendremos que definir una nueva clase que herede de EvolutionaryAlg y que implemente el ciclo reproductor del nuevo EA deseado. Una vez hecho esto, podremos utilizar en nuestro nuevo EA los modelos ya implementados en EvolutionaryAlg, como el multi-objetivo, el adaptativo o el jerárquico, sin ningún esfuerzo.

El GA celular está implementado con la clase CellularGA, que hereda las propiedades de GenGA (ya que en ambos casos la población completa se actualiza en cada generación) y establece la estructura de vecindarios dentro de la población. En concreto, para poder implementar un cGA en la clase CellularGA necesitamos extender GenGA con:

- **Una población estructurada.** Population no establece estructura alguna entre los individuos de la población, que es simplemente una lista de individuos sin ningún orden concreto. Por tanto, para el modelo celular ha sido necesario implementar una nueva clase, llamada PopGrid, que hereda de Population y que establece una estructura sobre la lista de individuos de Population de forma que a cada individuo se le asigna una posición de una malla bidimensional. En nuestra biblioteca es fácil utilizar otras topologías para la población, como las topologías de mundo pequeño, que se han propuesto recientemente para cGAs en algunos trabajos [118, 227]. Para ello basta con crear una nueva clase que herede de Population y que establezca la relación deseada entre los individuos.

- **Un vecindario.** La clase Neighborhood se ha implementado para obtener el vecindario de un individuo dado. Existen actualmente tres clases que heredan de Neighborhood para implementar tres de los vecindarios más conocidos. Estas clases son Linear5, Compact9 y Compact13 para los vecindarios L5, C9 y C13, respectivamente (consulte el Capítulo 2 para más detalles sobre los vecindarios). Añadir un nuevo vecindario es tan sencillo como crear una nueva clase que herede de Neighborhood y que establezca la estructura de vecindario deseada.

- **Orden de visita de los individuos.** Actualmente, existen varias políticas de actualización (ver Sección 2.5.1). El uso de una actualización síncrona o asíncrona se establece en una variable de la clase EvolutionaryAlg, mientras que el orden en que se recorren los individuos se toma de la clase abstracta CellUpdate. Existen en JCell actualmente 4 políticas de actualización distintas: la de barrido lineal (LineSweep), barrido aleatorio fijo (FixedRandomSweep), barrido aleatorio nuevo (NewRandomSweep), o elección uniforme (UniformChoice). La distinción entre estas cuatro políticas sólo tiene sentido en el caso asíncrono, puesto que en el caso síncrono todos los individuos se actualizan a la vez. Por tanto, se puede configurar el cGA síncrono con cualquiera de las cuatro políticas, con excepción de la elección uniforme puesto que, como se comentó en la Sección 2.5.1, con esta política es posible que no se actualicen todos los individuos en una generación.

Existe además una clase estática llamada Target que contiene una variable lógica, llamada maximize, para almacenar si el problema a resolver por el algoritmo requiere maximizar o minimizar la solución.

Además, esta clase se utiliza para realizar todas las comparaciones entre individuos, decidiendo en términos del valor de la variable maximize cuál representa una mejor solución del problema. Así, todas las comparaciones entre individuos y valores de fitness son una *caja negra* para el resto de las clases de JCell, de forma que estas clases llaman al mismo método de Target independientemente de si se está maximizando o minimizando, si consideramos restricciones o no, o incluso si estamos resolviendo un problema mono-objetivo o multi-objetivo. En este último caso, los términos mejor o peor no deben ser directamente aplicados, sino que en su lugar se utilizan en las comparaciones realizadas dentro de Target los conceptos de individuo dominante (considerado como mejor), dominado (peor), y no dominado (ni mejor ni peor).

12.2. Utilización de JCell

La forma más sencilla de configurar los parámetros de un EA cuando trabajamos con la biblioteca JCell consiste en establecer todos los parámetros deseados en un simple fichero de configuración. Esta opción, disponible gracias a la clase ReadConf, tiene claras ventajas, ya que el usuario no necesita tener conocimiento de los detalles internos de la codificación de JCell para utilizar la biblioteca, tales como los nombres de las variables o su tipo de datos. Además, el uso de un fichero de configuración nos permite cambiar de una forma simple y rápida la parametrización de nuestro EA, así como el propio algoritmo, sin necesidad de recompilar el código. Estas características hacen a nuestra biblioteca accesible a aquellos investigadores que pertenecen a cualquier otro campo, y que no quieren renunciar a la utilización de una técnica de optimización de última generación para sus investigaciones, ya que no es necesario tener ningún tipo de conocimiento de lenguajes de programación para utilizar JCell. Lo único que deben hacer es editar un fichero de configuración para establecer la parametrización y luego ejecutar el fichero `JCell.class` para lanzar el algoritmo.

Por tanto, utilizando este tipo de fichero de configuración, la funcionalidad y la facilidad de uso de JCell se ven fuertemente reforzadas. Y este es un aspecto realmente importante para su éxito y aceptación en la comunidad científica. El fichero de configuración se compone de pares:

$$\text{Nombre de Parámetro} \quad = \quad \text{Valor de Parámetro} \ .$$

En las figuras 12.2 y 12.3 mostramos dos ejemplos de ficheros de configuración para JCell. Como se puede ver, las líneas que comienzan por el carácter # son comentarios. El fichero mostrado en la Figura 12.2 es una configuración típica para un cGA (`Algorithm = cellular`), en el que el orden de visita de los individuos durante el ciclo reproductivo está definido por la política asíncrona *Fixed Random Sweep* (`UpdatePolicy = Asynchronous FRS`), y la forma de la población está establecida en 20×20 individuos (`Population = (20, 20)`). La evolución de los individuos en la población será mostrada por pantalla durante la ejecución (`ShowDisplay = true`), siendo actualizada cada generación, y JCell imprimirá por la salida estándar algunas trazas de información relevante de la evolución de la población durante la ejecución (`Verbose = true`). El problema que nos proponemos resolver es el problema de optimización combinatoria ECC (`Problem = problems.Combinatorial.ECC`), cuya clase está codificada en el paquete Java `problems.Combinatorial` de JCell (ver Apéndice A para más detalles sobre este problema). La condición de terminación del algoritmo consiste en encontrar la solución óptima al problema (que viene dada por la propia clase que implementa el problema), o alcanzar un máximo de 1 millón de evaluaciones de función de fitness (`EvaluationsLimit = 1000000`). El cromosoma de

```
##########################################
### FICHERO DE CONFIGURACIÓN DE JCELL  ###
###  Un cGA asíncrono (FRS) para ECC     ###
##########################################

## Tipo de algoritmo
Algorithm = cellular

## Política de actualización de individuos
UpdatePolicy = Asynchronous FRS

## Forma de la población (x, y)
Population = (20, 20)

## ¿Mostrar gráficamente la población?
ShowDisplay = true

## ¿Mostrar información durante la ejecución?
Verbose = true

## Problema a resolver
Problem = problems.Combinatorial.ECC

## Máximo número de evaluaciones permitido
EvaluationsLimit = 1000000

## Tipo de individuos
Individual = jcell.BinaryIndividual

## Define el vecindario a utilizar
Neighborhood = jcell.Linear5

## Política de selección
SelectionParent1 = jcell.CenterSelection
SelectionParent2 = jcell.TournamentSelection

## Probabilidad de recombinación
CrossoverProb = 1.0

## Operador de recombinación
Crossover = operators.Dpx

## Probabilidad de mutación de un individuo
MutationProb = 1.0

## Operador de mutación
Mutation = operators.BinaryMutation

## Política de reemplazo
Replacement = jcell.ReplaceIfNonWorse
```

Figura 12.2: Ejemplo de fichero de configuración para resolver ECC con un cGA asíncrono.

```
##########################################
## FICHERO DE CONFIGURACIÓN DE JCELL ##
## Un ssGA simple multi-objetivo      ##
## para el problema Fonseca           ##
##########################################

## Tipo de algoritmo
Algorithm = steady-state

## Tamaño de la población
Population = 400

## ¿Mostrar información durante la ejecución?
Verbose = true

## Problema a resolver
Problem = problems.MO.Fonseca

## Máximo número de evaluaciones permitido
EvaluationsLimit = 25000

## Tipo de individuos
Individual = jcell.RealIndividual

## Política de selección
SelectionParent1 = jcell.TournamentSelection
SelectionParent2 = jcell.TournamentSelection

## Probabilidad de recombinación
CrossoverProb = 1.0

## Operador de recombinación
Crossover = operators.SBX

## Probabilidad de mutación de un individuo
MutationProb = 1.0

## Operador de mutación
Mutation = operators.NonUniformMutation

## Política de reemplazo
Replacement = jcell.ReplaceIfNonWorse
```

Figura 12.3: Ejemplo de fichero de configuración para resolver el problema multi-objetivo Fonseca con un GA de estado estacionario.

individuos debe estar compuesto por genes binarios para resolver problemas de optimización combinatoria (`Individual = jcell.BinaryIndividual`), y la estructura de vecindario que queremos utilizar es Linear5 (`Neighborhood = jcell.Linear5`), de donde uno de los padres será seleccionado por torneo binario (`SelectionParent2 = jcell.TournamentSelection`). El otro padre (que será seleccionado en primer lugar) es el propio individuo actual (`SelectionParent1 = jcell.CenterSelection`). Los dos padres seleccionados no pueden ser en ningún caso el mismo individuo en JCell. El operador de recombinación de dos puntos (`Crossover = operators.Dpx`) se aplica a estos dos padres con probabilidad 100 % (`CrossoverProb = 1.0`), y el descendiente recientemente generado se somete siempre a mutación (`MutationProb = 1.0`) utilizando el operador de mutación binaria (`Mutation = operators.BinaryMutation`). Los genes del cromosoma son mutados con probabilidad 1/L, siendo L la longitud del cromosoma. Finalmente, el descendiente reemplaza al individuo actual en la población sólo si tiene un valor de fitness mejor o igual (`Replacement = jcell.ReplaceIfNonWorse`).

En la Figura 12.3 podemos ver un ejemplo de lo sencillo que resulta ejecutar un GA multi-objetivo de estado estacionario con JCell simplemente cambiando unos pocos parámetros del fichero de configuración explicado anteriormente (visto en la Figura 12.2). Al cambiar la clave `Algorithm` a `steady-state` obtenemos un GA de estado estacionario. La población en un algoritmo panmíctico (tanto de estado estacionario como generacional) es un conjunto de individuos sin estructura ni forma concreta. Por tanto, establecemos en este caso directamente el tamaño de la población en el fichero en lugar de indicar su forma (`Population = 400`). El problema que queremos afrontar es el llamado problema de Fonseca (`Problem = problems.MO.Fonseca`), definido en el Apéndice A, y el máximo número de evaluaciones de fitness está fijado en 25000, como es usual en el campo de optimización multi-objetivo. El problema Fonseca requiere la codificación de genes de tipo Double para representar las variables (`Individual = jcell.RealIndividual`), y por tanto necesitamos utilizar operadores de recombinación y mutación adecuados para este tipo de representación del cromosoma: *simulated binary* (`Crossover = operators.SBX`) y mutación no uniforme (`Mutation = operators.NonUniformMutation`), respectivamente. Por último, no hace falta especificar en este caso el vecindario (si se especificara se ignoraría) ya que la población no se encuentra estructurada en el caso del GA de estado estacionario.

Para terminar esta sección, se explican brevemente a continuación los principales parámetros que pueden utilizarse en el fichero de configuración de JCell. En la Tabla 12.2 mostramos algunos de los parámetros más importantes que nos permite la configuración de JCell, junto con los valores que se le pueden asignar, y una breve explicación de su significado. Ejemplos de estos parámetros son el tipo de algoritmo a utilizar, las probabilidades de recombinación y mutación, los operadores de selección de los padres, de recombinación o de mutación, o la política de reemplazo. Como ya se dijo anteriormente, la forma de configurar un parámetro consiste en escribir en el fichero de configuración, para el parámetro deseado, el nombre que aparece en la columna "parámetro", un símbolo "=", y uno de los posibles valores que aparecen en la columna "valores".

Los diversos tipos de problemas, así como los operadores de recombinación y mutación disponibles en JCell se muestran en las tablas 12.3, 12.4, y 12.5. Las tablas 12.3 y 12.4 muestran los operadores de recombinación y mutación disponibles en JCell, respectivamente, junto con el nombre del operador que representan y algunos comentarios, así como parámetros de configuración específicos de cada operador, en el caso de que tengan. El valor que se les asigna en las dos tablas a estos parámetros específicos de los operadores es el que toman por defecto. En la Tabla 12.5, para cada problema, se muestra el paquete de Java en el que se encuentran, algunos comentarios sobre el problema, como el tipo de codificación que utilizan, y una referencia de la sección de esta tesis en la que se puede encontrar una completa descripción del problema.

Tabla 12.2: Principales parámetros de configuración de JCell.

Parámetro	**Valores**	**Comentarios**
Algorithm	`cellular`	GA celular
	`generational`	GA generacional
	`steady-state`	GA de estado estacionario
UpdatePolicy	`Synchronous`	cGA síncrono
	`Asynchronous LS`	cGA asíncrono. Política Line Sweep
	`Asynchronous FRS`	cGA asíncrono. Política Fixed Random Sweep
	`Asynchronous NRS`	cGA asíncrono. Política New Random Sweep
	`Asynchronous UC`	cGA asíncrono. Política Uniform Choice
Population	`(x,y)`	cGA con población de $(x \times y)$
	`N`	GA panmíctico con población de tamaño `N`
ShowDisplay	`true`	Muestra gráficamente la evolución de la población
	`false`	No muestra gráficamente la evolución de la población
Verbose	`true`	Muestra por la salida estándar trazas de la ejecución
	`false`	No muestra por la salida estándar trazas de la ejecución
EvaluationsLimit	`N`	Establece el límite máximo de evaluaciones permitidas a `N`
Neighborhood	`jcell.Linear5`	El vecindario del cGA es L5
	`jcell.Compact9`	El vecindario del cGA es C9
	`jcell.Compact13`	El vecindario del cGA es C13
HierarchycalPop	`true`	Población jerárquica (Capítulo 7)
	`false`	Población no jerárquica
AdaptivePop	`adaptiveCGA.AF`	cGA con población adaptativa. Medida de diversidad: AF (Capítulo 6)
	`adaptiveCGA.PH`	cGA con población adaptativa. Medida de diversidad: PH
	`adaptiveCGA.AFPH`	cGA con población adaptativa. Medida de diversidad: AF+PH
CrossoverProb	`N`	La probabilidad de recombinación es: `N`
MutationProb	`N`	La probabilidad de mutar un individuo es: `N`
SelectionParent1 y SelectionParent2	`jcell.TournamentSelection`	Selección por torneo binario
	`jcell.RouleteWheelSelection`	Selección por ruleta
	`jcell.CenterSelection`	Selección del individuo actual
	`jcell.BestSelection`	Selección del mejor individuo del vecindario
	`jcell.LinearRankSelection`	Selección por ordenación lineal
Replacement	`jcell.ReplaceIfNonWorse`	El descendiente reemplaza al individuo seleccionado si no es peor
	`jcell.ReplaceIfBetter`	El descendiente reemplaza al individuo seleccionado si es mejor
	`jcell.ReplaceAlways`	El descendiente reemplaza al individuo seleccionado siempre
ArchiveManagement	`CMoEA.Crowding`	Crowding para el archivo de soluciones no dominadas
	`CMoEA.AdaptiveGrid`	Grid Adaptativo para gestionar el archivo de soluciones no dominadas
Crossover	ver Tabla 12.3	Operador de recombinación
Mutation	ver Tabla 12.4	Operador de mutación
Problem	ver Tabla 12.5	Problema a resolver

12.3. Conclusiones

En este capítulo presentamos y describimos la biblioteca desarrollada durante las investigaciones contenidas en esta tesis. JCell, que es como se llama esta biblioteca, es, por sí misma, un resultado importante de este trabajo de tesis, puesto que pone al alcance de cualquier investigador una potente herramienta de optimización que puede utilizar en sus estudios. Además, JCell permite al investigador especializado en el campo de los algoritmos evolutivos profundizar con muy poco esfuerzo en los estudios aquí realizados, desarrollar nuevos estudios basados en algoritmos evolutivos celulares, o incluso exportar las nuevas ideas propuestas en esta tesis a otros campos.

Tabla 12.3: Operadores de recombinación disponibles en la biblioteca JCell.

Clase	Nombre de operador	Comentarios y parámetros
operators.Ax	Cruce asimétrico	Para individuos de genotipo real `pBiasAX` = 0.5
operators.BLXalpha	Cruce de aleación	Para individuos de genotipo real; `AlphaBLX` = 0.5
operators.Dpx	Cruce de dos puntos	Para todos los tipos de individuos
operators.DX	Cruce discreto	Para individuos de genotipo real
operators.ELX	Cruce de línea extendida	Para individuos de genotipo real
operators.Erx	Cruce por recombinación de ejes	Para individuos de genotipo permutación de enteros
operators.FX	Cruce plano	Para individuos de genotipo real
operators.LBGAX	Cruce por BGA lineal	Para individuos de genotipo real
operators.LX	Cruce lineal	Para individuos de genotipo real
operators.OX	Cruce ordenado	Para individuos de genotipo permutación de enteros
operators.PBX	Cruce centrado en los padres	Para individuos de genotipo real `AlphaPBX` = 0.8
operators.Pmx	Cruce por correspondencia parcial	Para individuos de genotipo permutación de enteros
operators.PNX	Cruce normal centrado en los padres	Para individuos de genotipo real `LambdaPNX` = 0.35 `RoPNX` = 2.0
operators.Px	Cruce probabilístico	Para todos los tipos de individuos `pBiasPX` = 0.5
operators.SBX	Cruce binario simulado	Para individuos de genotipo real `distributionIndexSBX` = 20.0
operators.Spx	Cruce de un punto	Para todos los tipos de individuos
operators.WHX	Cruce del heurístico de Wright	Para individuos de genotipo real

Tabla 12.4: Operadores de mutación disponibles en la biblioteca JCell.

Clase	Nombre de operador	Comentarios y parámetros
operators.BinaryMutation	Mutación binaria	Para individuos de genotipo binario
operators.CauchyMutation	Mutación de Cauchy	Para individuos de genotipo real `deviationCM` = 1.0
operators.FloatUniformMutation	Mutación uniforme	Para individuos de genotipo real
operators.FloatNonUniformMutation	Mutación no uniforme	Para individuos de genotipo real
operators.GaussianMutation	Mutación gaussiana	Para individuos de genotipo real `deviationGM` = 1.0
operators.MuhlenbeinMutation	Mutación de Mühlenbein	Para individuos de genotipo real
operators.PolynomialMutation	Mutación polinómica	Para individuos de genotipo real `distributionIndexPM` = 20.0

Tabla 12.5: Problemas incluidos en la biblioteca JCell.

Clase	Comentarios	Referencia
problems.MO.Constr_Ex	Multi-objetivo; Codificación Real	Sección A.3
problems.MO.Fonseca	Multi-objetivo; Codificación Real	Sección A.3
problems.MO.Golinski	Multi-objetivo; Codificación Real	Sección A.3
problems.MO.Kursawe	Multi-objetivo; Codificación Real	Sección A.3
problems.MO.Schaffer	Multi-objetivo; Codificación Real	Sección A.3
problems.Combinatorial.COUNTSAT	Codificación Binaria	Sección A.1.1
problems.Combinatorial.ECC	Codificación Binaria	Sección A.1.2
problems.Combinatorial.FMS	Codificación Binaria; Variables codificadas con 32 bits	Sección A.1.3
problems.Combinatorial.MAXCUT	Codificación Binaria; Instancias 100, 20.01 y 20.09	Sección A.1.5
problems.Combinatorial.MMDP	Codificación Binaria	Sección A.1.6
problems.Combinatorial.MTTP	Codificación Binaria; Instancias de 20, 100, y 200 variables	Sección A.1.7
problems.Combinatorial.OneMax	Codificación Binaria; 500 variables	Sección A.1.8
problems.Combinatorial.PEAK	Codificación Binaria	Sección A.1.10
problems.Continuous.Ackley	Codificación Real	Sección A.2.1
problems.Continuous.Chebyschev	Codificación Real	Sección A.2.2
problems.Continuous.EF10	Codificación Real	Sección A.2.1
problems.Continuous.FMS	Codificación Real	Sección A.2.2
problems.Continuous.Griewangk	Codificación Real	Sección A.2.1
problems.Continuous.Rastrigin	Codificación Real	Sección A.2.1
problems.Continuous.Rosenbrock	Codificación Real	Sección A.2.1
problems.Continuous.Schwefel	Codificación Real	Sección A.2.1
problems.Continuous.Sle	Codificación Real	Sección A.2.2
problems.Continuous.Sphere	Codificación Real	Sección A.2.1

Parte III

Aplicaciones de los Algoritmos Genéticos Celulares

Capítulo 13

Caso Discreto: El Problema de Rutas de Vehículos

El transporte juega un papel importante en las tareas de logística de muchas compañías, ya que normalmente representa un alto porcentaje del valor añadido a los bienes. Por tanto, la utilización de métodos computacionales en el transporte suele aportar ahorros de más del 20 % del coste total (véase el Capítulo 1 en [275]).

Un problema destacado en el campo del transporte consiste en encontrar la ruta óptima para una flota de vehículos que sirven a un conjunto de clientes. En este problema, un conjunto arbitrario de clientes deben recibir artículos de un almacén central. Este escenario general presenta muchas situaciones para definir escenarios de problemas relacionados, como por ejemplo: determinar el número óptimo de vehículos, encontrar las rutas más cortas, etc., todos ellos están sujetos a muchas restricciones como la velocidad del vehículo, el tiempo de ventana para las entregas, etc. Esta variedad de escenarios nos lleva a una plétora de distintos problemas en la práctica. En los Capítulos 10 y 14 de [275] podemos encontrar algunos casos de estudio de referencia donde la aplicación de los algoritmos para calcular las rutas de vehículos de mínimo coste nos aportan ahorros sustanciales.

Como hemos dicho antes, el Problema de Rutas de Vehículos (VRP) [68] consiste en entregar artículos, utilizando las rutas de coste mínimo, a un determinado conjunto de clientes (de los que conocemos sus demandas), comenzando y acabando en el almacén, como puede verse en las Figuras 13.1a y 13.1b (se muestra una descripción detallada del problema en la Sección 13.1).

El VRP es una fuente muy importante de problemas, ya que resolverlo es equivalente a resolver múltiples problemas TSP a la vez. Debido a la dificultad de este problema (NP-duro) y a sus múltiples aplicaciones industriales, ha sido ampliamente estudiado tanto de manera práctica como teórica [58]. Hay un gran número de extensiones del VRP canónico. Una extensión básica se conoce como VRP con Capacidad limitada –CVRP–, en el que los vehículos tienen capacidades fijas de un único artículo. Se pueden construir muchas variantes a partir de CVRP; algunas de las más importantes [275] son aquellas que incluyen restricciones en las ventanas de tiempos –VRPTW– (los clientes deben ser abastecidos siguiendo un determinado esquema de tiempos), con recogida y entrega de artículos –VRPPD– (los clientes demandan artículos para que sean recogidos o entregados), o con recogida de artículos sobrantes –VRPB– (como VRPPD, pero se deben completar las entregas antes de que se realice ninguna recogida).

En realidad, hay muchas más extensiones de este problema, como el uso de múltiples almacenes (MDVRP), entregas compartidas por más de un vehículo (SDVRP), variables estocásticas (SVRP), o programación periódica (PVRP). El lector puede encontrar todos ellos en nuestro servidor web público, junto con las últimas mejores soluciones conocidas, publicaciones y más material relacionado [84].

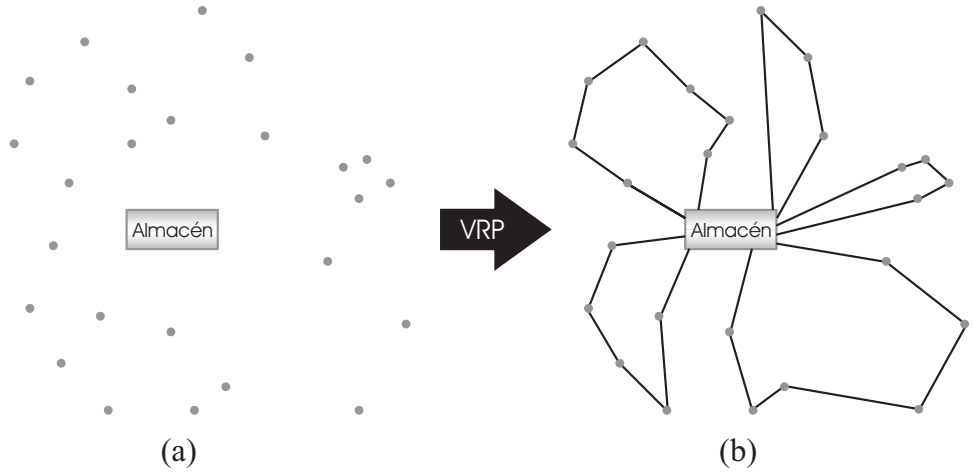


Figura 13.1: El Problema de Rutas de Vehículos consiste en abastecer a un conjunto de clientes geográficamente distribuidos (puntos) desde un almacén (a) utilizando las rutas de coste mínimo (b).

En este capítulo consideramos el Problema de Rutas de Vehículos con Capacidad limitada (CVRP), en el que una flota fija de vehículos de reparto que tienen la misma capacidad, deben abastecer las conocidas demandas de los clientes de un único artículo, desde un almacén común con el mínimo coste de tránsito (longitud recorrida). El CVRP ha sido estudiado en un gran número de trabajos distintos de la literatura, pero (hasta donde conocemos) ningún trabajo considera un conjunto tan grande de instancias del problema, ya que esto significa resolver 160 instancias distintas. Nosotros usaremos este enorme conjunto de problemas para probar el comportamiento de nuestro algoritmo en distintos escenarios, y así realizar un profundo análisis de nuestro algoritmo y una vista general de este problema que no dependa de una selección ad hoc de instancias individuales. Las instancias incluidas están caracterizadas por diferentes aspectos: pertenecientes al mundo real, teóricas, con clientes agrupados por zonas o no, con demandas de clientes tanto homogéneas como heterogéneas, con tiempos de entrega o sin ellos, etc.

En la historia reciente del VRP ha habido una evolución constante en la calidad de las metodologías resolutivas utilizadas en este problema, pertenecientes tanto al campo de investigación exacto como al heurístico. De todas formas, dada la dificultad del problema, ningún método exacto conocido es capaz de encontrar el óptimo para instancias que contengan más de 50 clientes [125, 275]. En realidad, está claro que, como para la mayoría de los problemas complejos, las heurísticas genéricas (no desarrolladas para ningún problema en particular) no pueden competir en la calidad de la solución con las técnicas actuales como las que están descritas en [63, 275]. Además, la potencia de la exploración de algunas técnicas modernas como los Algoritmos Genéticos o los Sistemas de Hormigas no está explorada a fondo todavía, especialmente cuando se combinan con una etapa de búsqueda local eficaz. Todas estas consideraciones podrían permitirnos refinar las soluciones hacia el óptimo. Particularmente, en este capítulo se presenta un Algoritmo Genético Celular (cGA) [195] hibridado con unos operadores de recombinación y mutación especializados y con un algoritmo de búsqueda local para resolver CVRP.

La contribución de este capítulo es por tanto definir un potente pero sencillo cMA (cGA altamente hibridizado) capaz de competir con las mejores aproximaciones conocidas en la resolución de CVRP en términos de precisión (coste final) y esfuerzo (número de evaluaciones realizadas). Con este fin, probamos nuestro algoritmo con una gran selección de instancias (160), que nos permitirá garantizar conclusiones significativas. Por otra parte, comparamos nuestro resultado con los mejores existentes y en algunos casos incluso los mejoramos. El lector puede encontrar en [14] trabajos preliminares realizando una comparación entre nuestro algoritmo y otras heurísticas conocidas para un conjunto reducido de 8 instancias. En ese trabajo mostramos las ventajas de insertar técnicas de búsqueda local en un cGA para resolver el CVRP, ya que nuestro cGA híbrido resultó ser el mejor de todos los comparados en términos de precisión y tiempo. Los GAs celulares representan un paradigma mucho más sencillo que otros como la Búsqueda Tabu (TS) [114, 276] y algoritmos similares (muy abstractos o especializados) [42, 233] si tenemos en cuenta la comprensión y la implementación. Éste también es un punto importante, ya que un mayor énfasis en la sencillez y flexibilidad es hoy en día un deber en la investigación para obtener unas contribuciones realmente útiles [63].

El capítulo está organizado de la siguiente manera. En la Sección 13.1 definimos CVRP. El cGA híbrido que hemos propuesto se describe en la Sección 13.2. El objetivo de la Sección 13.3 es justificar las elecciones que hemos tomado para el diseño de nuestro algoritmo altamente hibridado. En la Sección 13.4 se presentan los resultados de nuestras pruebas y las comparamos con los mejores valores conocidos de la literatura. Destacamos en la Sección 13.5 las nuevas mejores soluciones encontradas en este estudio. Finalmente, nuestras conclusiones y nuestras líneas de investigación futuras se presentan en la Sección 13.6.

13.1. Problema de Rutas de Vehículos

El VRP se puede definir como un problema de programación de números enteros perteneciente a la categoría de problemas NP-duros [180]. Entre las diferentes variedades de VRP nosotros trabajaremos con el VRP de Capacidad limitada (CVRP), en el que cada vehículo tiene una capacidad uniforme de un único artículo. Definimos el CVRP sobre un grafo no dirigido $G = (\vec{V}, \vec{E})$ donde $\vec{V} = \{v_0, v_1, \ldots, v_n\}$ es un conjunto de vértices y $\vec{E} = \{(v_i, v_j)/v_i, v_j \in \vec{V}, i < j\}$ es un conjunto de ejes.

Los vértices v_0 parten del *almacén*, y es desde donde m vehículos idénticos de capacidad Q deben abastecer a todas las *ciudades* o *clientes*, representados por un conjunto de n vértices $\{v_1, \ldots, v_n\}$. Definimos en E una matriz $C = (c_{ij})$ de *coste*, *distancia* o *tiempo de viaje* no negativos entre los clientes v_i y v_j. Cada cliente v_i tiene una demanda no negativa de artículos q_i y tiempos de entrega δ_i (tiempo necesario para descargar todos los artículos). Siendo $\vec{V_1}, \ldots, \vec{V_m}$ una partición de $\vec{V}$, una ruta $\vec{R_i}$ es una permutación de los clientes en $\vec{V_i}$ especificando el orden en el que se visitan, comenzando y terminando en el almacén v_0. El coste de una ruta dada $\vec{R_i} = \{v_0, v_1, \ldots, v_{k+1}\}$, donde $v_j \in \vec{V}$ y $v_0 = v_{k+1} = 0$ (0 indica el almacén), viene dada por:

$$\text{Cost}(\vec{R_i}) = \sum_{j=0}^{k} c_{j,j+1} + \sum_{j=0}^{k} \delta_j \ , \tag{13.1}$$

y el coste de la solución al problema ($\vec{S}$) es:

$$F_{\text{CVRP}}(\vec{S}) = \sum_{i=1}^{m} \text{Cost}(\vec{R_i}) \ . \tag{13.2}$$

El CVRP consiste en determinar un conjunto de m rutas (i) de coste total mínimo –como especifica la Ecuación 13.2–; (ii) empezando y terminando en el almacén v_0; de forma que (iii) cada cliente es visitado una sola vez por un sólo vehículo; sujeto a las restricciones (iv) de que la demanda total de cualquier ruta no exceda Q ($\sum_{v_j \in \vec{R}_i} q_j \leq Q$); y (v) la duración total de cualquier ruta no supera el límite preseleccionado D ($\text{Cost}(\vec{R}_i) \leq D$). Todos los vehículos tienen la misma capacidad y transportan el mismo tipo de artículo. El número de vehículos puede ser un valor de entrada o una variable de decisión. En este estudio, la longitud de las rutas se minimiza independientemente del número de vehículos utilizados.

Queda claro tras nuestra descripción que el VRP está fuertemente relacionado con dos difíciles problemas combinatorios. Por un lado, podemos tener una instancia de Problema Múltiple del Viajante de Comercio (*Multiple Travelling Salesman Problem* o MTSP) con sólo establecer $C = \infty$. Una instancia del MTSP se puede transformar en una instancia del TSP añadiéndole al grafo $k-1$ copias adicionales del nodo 0 (el almacén) y de sus ejes incidentes (no existen ejes entre los k nodos de almacén). Por otro lado, la cuestión de si existe una solución factible para una instancia dada del VRP es una instancia del Problema de Llenado de Recipiente (*Bin Packing Problem* o BPP). Por tanto, el VRP es un problema extremadamente difícil de resolver en la práctica debido a la interacción de estos dos modelos tan complejos (MTSP y BPP). En realidad, las instancias más grandes del VRP que se han podido resolver hasta el momento tienen dos órdenes de magnitud menos que la de TSP [234].

13.2. El Algoritmo Propuesto

En esta sección vamos a presentar una descripción muy breve del algoritmo que hemos desarrollado (para más información sobre el funcionamiento de cGAs consulte el Capítulo 2). En el Algoritmo 13.1 podemos observar el pseudocódigo de JCell2o1i, el cGA híbrido que proponemos en este estudio. Básicamente, utilizamos un sencillo cGA canónico altamente hibridado con operadores de mutación y recombinación específicos, y también con una etapa añadida de post-optimización local (línea 12). Este método de búsqueda local consiste en aplicar el método *2-Opt*, tras lo que a la mejor solución obtenida se le aplica *1-Intercambio*. Estos dos métodos son algoritmos de optimización local bien conocidos en la literatura (están descritos en detalle en la Sección 13.2.4).

Calculamos el valor de fitness que se asocia a cada individuo de la siguiente forma [14, 114]:

$$f(\vec{S}) = F_{\text{CVRP}}(\vec{S}) + \lambda \cdot \text{penalCap}(\vec{S}) + \mu \cdot \text{penalTiempo}(\vec{S}) \,, \tag{13.3}$$

$$f_{\text{eval}}(\vec{S}) = f_{\max} - f(\vec{S}) \,. \tag{13.4}$$

El objetivo de nuestro algoritmo consiste en maximizar $f_{\text{eval}}(\vec{S})$ (Ecuación 13.4) minimizando $f(\vec{S})$ (Ecuación 13.3). El valor $f_{\max}$ debe ser mayor o igual que la peor solución posible al problema que no incumpla ninguna restricción. La función $f(\vec{S})$ se calcula añadiendo los costes totales de todas las rutas ($F_{\text{CVRP}}(\vec{S})$ –ver (Ecuación 13.2)–), y penaliza el valor de fitness sólo en el caso de que se excedan la capacidad de cualquier vehículo y/o el tiempo de cualquier ruta. Las funciones 'penalCap($\vec{S}$)' y 'penalTiempo($\vec{S}$)' devuelven una cuantificación del exceso de la solución en capacidad y tiempo, respectivamente, con respecto al máximo valor permitido para cada ruta. Estos valores devueltos por 'penalCap($\vec{S}$)' y 'penalTiempo($\vec{S}$)' son ponderados con valores λ y μ, respectivamente. En este trabajo, hemos utilizado los valores $\lambda = \mu = 1000$ [90].

En las secciones 13.2.1 a 13.2.4 explicamos en detalle los principales aspectos que caracterizan nuestro algoritmo (llamado JCell2o1i). Se puede aplicar el algoritmo con todos los operadores mencionados o con sólo algunos de ellos con el fin analizar su contribución separada en el rendimiento del algoritmo.

Algoritmo 13.1 Pseudocódigo de JCell2o1i.

```
1.  proc Evoluciona(cga)        //Parámetros del algoritmo en 'cga'
2.  GeneraPoblaciónInicial(cga.pobl);
3.  Evaluación(cga.pobl);
4.  mientras !CondiciónParada() hacer
5.    para x ← 1 hasta ANCHO hacer
6.      para y ← 1 hasta ALTO hacer
7.        vecinos ← CalculaVecindario(cga,posición(x,y));
8.        padre1 ← IndividuoEn(cga,posición(x,y));
9.        padre2 ← TorneoBinario(vecinos);
10.       descendiente ← Recombinación(cga.Pc,padre1,padre2);
11.       descendiente ← Mutación(cga.Pm,descendiente);
12.       descendiente ← BúsquedaLocal(cga.Pl,descendiente);
13.       Evaluación(descendiente);
14.       InsertaSiNoPeor(posición(x,y),descendiente,cga,pobl_aux);
15.     fin para
16.   fin para
17.   cga.pop ← pobl_aux;
18.   ActualizaEstadísticas(cga);
19. fin mientras
20. fin proc Evoluciona;
```

13.2.1. Representación del Problema

En un GA, los individuos representan soluciones candidatas. Una solución candidata para una instancia de CVRP debe especificar el número de vehículos necesarios, la asignación de las demandas a todos estos vehículos y además el orden de entrega de cada ruta. Adoptamos en este trabajo una representación que consiste en una permutación de números enteros. Cada permutación contendrá tanto a los clientes como a los divisores de rutas (delimitando distintas rutas), así que usaremos una permutación de números enteros $[0 \ldots n-1]$ con una longitud $n = c + k$ para representar una solución para el CVRP con c clientes y un máximo de $k+1$ rutas posibles. Representamos a los clientes por números $[0 \ldots c-1]$, mientras que los divisores de rutas pertenecen al rango $[c \ldots n-1]$. Nótese que debido a la naturaleza de los cromosomas (permutación de números enteros) los divisores de ruta deben ser números distintos, aunque se podría utilizar el mismo número para diseñar divisores de ruta en el caso de usar otra posible configuración de cromosomas.

Cada ruta se compone de los clientes localizados entre cada dos divisores de ruta del individuo. Por ejemplo, en la Figura 13.2 esbozamos un individuo representando una posible solución para una hipotética instancia de CVRP con 10 clientes utilizando un máximo de 4 vehículos. Los valores $[0, \ldots, 9]$ representan a los clientes mientras que $[10, \ldots, 12]$ son los divisores de ruta. La **Ruta 1** comienza en el almacén, visita los clientes 4–5–2 (en ese orden), y vuelve al almacén. La **Ruta 2** va desde el almacén a los clientes 0–3–1 y vuelve. El vehículo de la **Ruta 3** comienza en el almacén y visita a los clientes 7–8–9. Finalmente, en la **Ruta 4**, sólo se visita al cliente 6 desde el almacén. Las rutas vacías se permiten en esta representación situando dos divisores de ruta consecutivamente, sin que haya ningún cliente entre ellos.

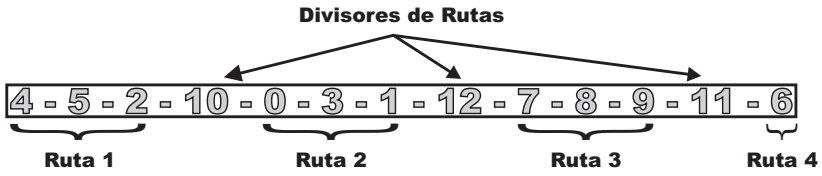


Figura 13.2: Individuo representando una solución para 10 clientes y 4 vehículos.

13.2.2. Recombinación

En los GAs se utiliza la recombinación como un operador que combina partes en dos (o más) patrones para intentar transmitir los buenos elementos básicos que contienen a su descendiente. El operador de recombinación que utilizamos es el operador de recombinación de ejes (ERX) [291], ya que ha sido ampliamente descrito como el más apropiado para trabajar con permutaciones en comparación con otros operadores generales como el de recombinación por orden (OX) o el de recombinación por emparejamientos parciales (PMX). ERX construye un descendiente conservando ejes de sus dos padres. Para ello, se utiliza una *lista de ejes*. Esta lista contiene, para cada ciudad, los ejes de los padres que empiezan o acaban en ella (véase Figura 13.3).

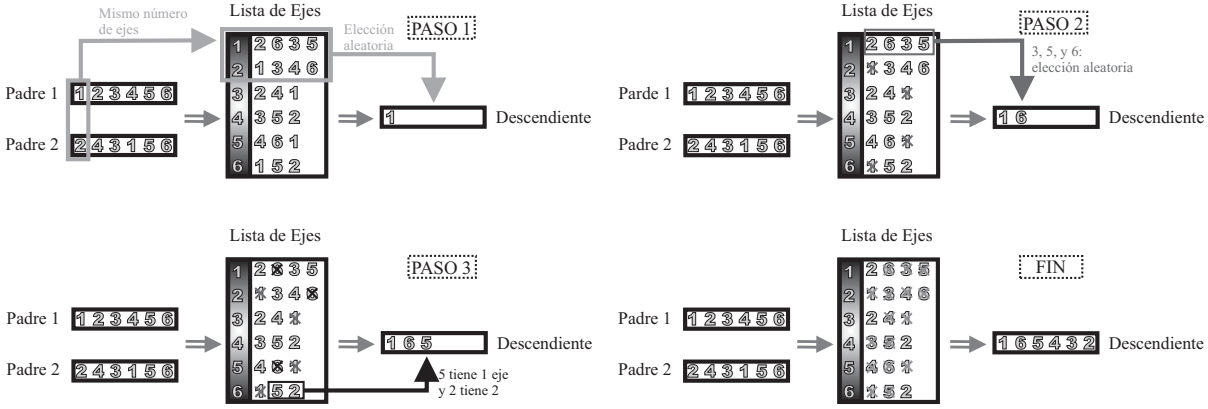


Figura 13.3: Operador de recombinación de ejes.

Después de construir la lista de ejes de los dos padres, el algoritmo ERX genera la solución hija de la siguiente manera. El primer gen del descendiente se escoge de entre el primer gen de cada padre. Específicamente, se escoge el gen con menor número de ejes. En caso de empate, escogeremos el primer gen del primer padre. El resto de genes se escogen tomando al vecino que tenga la lista de ejes más corta. Los empates se rompen eligiendo al primer cliente encontrado que satisfaga completamente este criterio de la menor lista. Una vez que hemos escogido un gen, lo eliminamos de la lista de ejes.

Algoritmo 13.2 El algoritmo de mutación.

```
1.  proc Mutación(pm, ind)      // 'pm' es la probabilidad de mutación, e 'ind' es el individuo a mutar
2.  para i ← 1 hasta ind.Longitud() hacer
3.    si rand0to1()<pm entonces
4.      r = rand0to1();
5.      si r < 0.33 entonces
6.        ind.Inversión(i, randomInt(ind.longitud()));
7.      en otro caso si r > 0.66 entonces
8.        ind.Inserción(i, randomInt(ind.Longitud()));
9.      en otro caso
10.       ind.Intercambio(i, randomInt(ind.Longitud()));
11.     fin si
12.   fin si
13. fin para
14. fin proc Mutación;
```

13.2.3. Mutación

El operador de mutación que usamos en nuestros algoritmos juega un papel importante durante la evolución ya que está encargado de introducir un grado de diversidad considerable en cada generación, contrarrestando de esta manera la gran presión de selección que ejerce el método de búsqueda local que vamos a utilizar. La mutación consiste en aplicar operaciones de *Inserción*, *Intercambio* o *Inversión* a cada gen con igual probabilidad (véase el Algoritmo 13.2).

Estos tres operadores de mutación (véase Figura 13.4) son métodos muy conocidos existentes en la literatura, y que se aplican normalmente en los problemas de rutas. Lo que se nos ocurrió fue unir estos tres en un nuevo operador *Combinado*. El operador de *Inserción* [101] selecciona un gen (un cliente o un divisor de rutas) y lo inserta en otro lugar elegido aleatoriamente del mismo individuo. El operador de *Intercambio* [35] consiste en elegir aleatoriamente dos genes de una solución e intercambiarlos. Finalmente, el operador de *Inversión* [150] coloca el orden de visita de los genes entre dos puntos de la permutación elegidos aleatoriamente en el orden inverso. Note que los cambios provocados se pueden dar tanto intra- como inter-ruta en los tres operadores. Descrito formalmente, dada una solución potencial $S = \{s_1, \ldots, s_{p-1}, s_p, s_{p+1}, \ldots, s_{q-1}, s_q, s_{q+1}, \ldots, s_n\}$, donde p y q son índices escogidos aleatoriamente, y n es la suma del número de clientes más el número de divisores de rutas ($n = c + k$), entonces la nueva solución S' obtenida tras aplicar cada uno de los mecanismos propuestos se muestra a continuación:

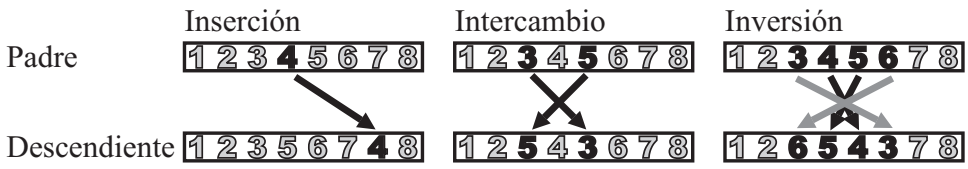


Figura 13.4: Las tres mutaciones utilizadas.

$$\text{Inserción:} \quad S' = \{s_1, \ldots, s_{p-1}, s_{p+1}, \ldots, s_{q-1}, s_q, s_p, s_{q+1}, \ldots, s_n\} \,, \tag{13.5}$$

$$\text{Intercambio :} \quad S' = \{s_1, \ldots, s_{p-1}, s_q, s_{p+1}, \ldots, s_{q-1}, s_p, s_{q+1}, \ldots, s_n\} \,, \tag{13.6}$$

$$\text{Inversión:} \quad S' = \{s_1, \ldots, s_{p-1}, s_q, s_{q-1}, \ldots, s_{p+1}, s_p, s_{q+1}, \ldots, s_n\} \,. \tag{13.7}$$

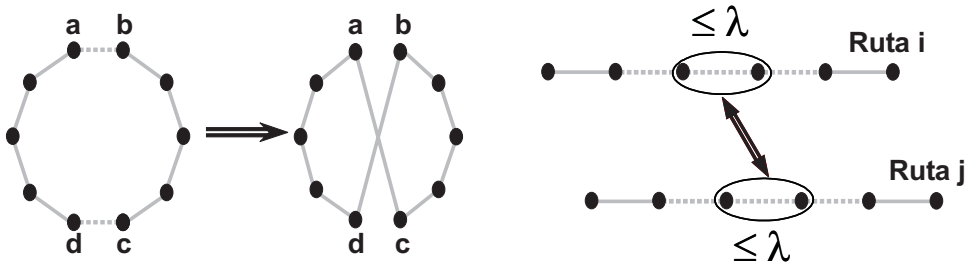
13.2.4. Búsqueda Local

Está muy claro, después de estudiar la literatura existente sobre VRP, que la utilización de un método de búsqueda local es casi obligatorio para obtener resultados de gran calidad [14, 42, 242]. Por eso nosotros nos centramos desde el principio en la aplicación de dos de las mejores técnicas existentes en los últimos años. Por tanto, añadiremos una etapa de refinamiento local a alguno de nuestros algoritmos, que consiste en aplicar optimización local 2-Opt [66] y 1-Intercambio [226] a cada individuo.

Por un lado, el método de búsqueda local 2-Opt trabaja dentro de cada ruta. Se seleccionan aleatoriamente dos ejes no adyacentes –(a, b) y (c, d)– de una sola ruta, los elimina, rompiendo por tanto la ruta en dos partes, y reconecta estas dos partes de la otra manera posible: (a, c) y (b, d) (Figura 13.5a). Por tanto, para una ruta dada $R = \{r_1, \ldots, r_a, r_b, \ldots, r_c, r_d, \ldots, r_n\}$, siendo (r_a, r_b) y (r_c, r_d) dos ejes no adyacentes seleccionados aleatoriamente, la nueva ruta R', obtenida tras aplicar el método 2-Opt a los dos ejes en cuestión, será $R' = \{r_1, \ldots, r_a, r_c, \ldots, r_b, r_d, \ldots, r_n\}$.

Por otro lado, el método de optimización local λ-Intercambio que usamos está basado en el análisis de todas las combinaciones posibles de un máximo de λ clientes entre conjuntos de rutas (Figura 13.5b). Por tanto, este método funciona en clientes que son cambiados de una ruta a otra o que son intercambiados entre rutas. Una solución para el problema se representa por un conjunto de rutas $S = \{R_1, \ldots, R_p, \ldots, R_q, \ldots, R_k\}$, donde R_i es el conjunto de clientes abastecidos en la ruta i. Nuevas soluciones vecinas se pueden obtener tras aplicar λ-Intercambio entre un par de rutas R_p y R_q; para ello reemplazamos cada conjunto de clientes $S_1 \subseteq R_p$ de tamaño $|S_1| \leq \lambda$ por cualquier otro $S_2 \subseteq R_q$ de tamaño $|S_2| \leq \lambda$. De esta manera, obtenemos dos rutas nuevas $R'_p = (R_p - S_1) \bigcup S_2$ y $R'_q = (R_q - S_2) \bigcup S_1$, que forman parte de la nueva solución $S' = \{R_1 \ldots R'_p \ldots R'_q \ldots R_k\}$.

Por tanto, 2-Opt busca mejores soluciones modificando el orden de visita de los clientes dentro de una ruta, mientras que el método λ-Intercambio funciona con clientes que son trasladados de una ruta a otra o que son intercambiados entre rutas. Nuestra etapa de búsqueda local se aplica a un individuo después de los operadores de recombinación y mutación, y devuelve la mejor solución entre las mejores encontradas tras aplicar 2-Opt y 1-Intercambio, o el actual en caso de que sea mejor (véase el pseudocódigo en el Algoritmo 13.3). En la etapa de búsqueda local, el algoritmo aplica 2-Opt a todos los pares de ejes no adyacentes en cada ruta y 1-Intercambio a todos los subconjuntos que como mucho tengan un cliente entre todos los pares de rutas.



(a) (b)

Figura 13.5: 2-Opt trabaja dentro de una ruta (**a**), mientras que λ-Intercambio afecta a dos rutas (**b**).

Algoritmo 13.3 La etapa de búsqueda local.

```
1. proc Búsqueda_Local(ind)        // 'ind' es el individuo a mejorar
2. // MAX_PASOS = 20
3. para s ← 0 hasta MAX_PASOS hacer
4.   //Primero: 2_Opt. K es el número de rutas
5.   mejor2_Opt = ind;
6.   para r ← 0 hasta K hacer
7.     sol = 2_Opt(ind,r);
8.     si Mejor(sol,mejor2_Opt) entonces
9.       mejor2_Opt = sol;
10.    fin si
11.  fin para
12. fin para
13. //Segundo: 1_Intercambio.
14. mejor1_Intercambio = ind;
15. para s ← 0 hasta MAX_PASOS hacer
16.   para i ← 0 hasta Longitud(ind) hacer
17.     para j ← i+1 hasta Longitud(ind) hacer
18.       sol = 1_Intercambio(i,j);
19.       si Mejor(sol,mejor1_Intercambio) entonces
20.         mejor1_Intercambio = sol;
21.       fin si
22.     fin para
23.   fin para
24. fin para
25. devolver Mejor(mejor2_Opt, mejor1_Intercambio);
26. fin proc Búsqueda_Local;
```

Resumiendo, el funcionamiento del JCell es bastante simple: en cada generación, se obtiene un descendiente para cada individuo tras aplicar el operador de recombinación (ERX) a sus dos padres (elegidos de su vecindario). Los descendientes se alteran con una mutación *combinada* especial, y después se le aplica una etapa de post-optimización local a los individuos mutados. Este algoritmo de búsqueda local consiste en aplicar al individuo dos métodos de búsqueda local (2-Opt y 1-Intercambio), y devuelve el mejor individuo de entre el individuo de entrada y la salida de la búsqueda local. La población de la próxima generación se compone de la actual después de reemplazar sus individuos por sus descendientes en caso de que no sean peores.

13.3. Buscando un Nuevo Algoritmo: El Camino hacia JCell2o1i

En esta sección vamos a presentar un estudio preliminar para justificar algunas elecciones realizadas en el diseño final de nuestro algoritmo, JCelll2o1i. Específicamente, en la Sección 13.3.1 ilustramos las ventajas obtenidas al utilizar una población descentralizada (celular) en vez de una panmíctica

Tabla 13.1: Parametrización usada en el algoritmo.

Tamaño de la Población	100 Individuos (10×10)
Selección de los Padres	Individuo Actual + Torneo Binario
Recombinación	ERX, $p_c = 0.65$
Mutación de Individuos	Inserción, Intercambio o Inversión (igual prob.), $p_m = 0.85$
Probabilidad de Mutación de Bit	$p_{bm} = 1.2/L$
Longitud del Individuo	L
Reemplazo	Reemp_si_no_Mejor
Búsqueda Local (LS)	2-Opt + 1-Intercambio, 20 Etapas de Optimización

(no estructurada). En la Sección 13.3.2 analizamos el comportamiento de tres algoritmos utilizando de manera separada tres métodos de mutación (muy utilizados en VRP), que justificarán más adelante nuestra propuesta de mutación *combinada*. Finalmente, justificaremos la elección del método de búsqueda local utilizado (formado por dos métodos muy conocidos) en la Sección 13.3.3. Las mejoras observadas en este estudio inicial nos llevan a nuestro cGA híbrido propuesto finalmente.

Para los estudios que estamos realizando, hemos escogido algunas instancias duras del conjunto de Christofides, Mingozzi y Toth [58]. Específicamente, utilizamos las tres instancias más pequeñas del conjunto con y sin restricciones de tiempos de entrega y longitud máxima de las rutas (`CMT-n51-k5`, `CMT-n76-k10` y `CMT-n101-k8`, y `CMTd-n51-k6`, `CMTd-n76-k11` y `CMTd-n101-k9`). Para obtener un conjunto más amplio (y por tanto, resultados más fiables), hemos seleccionado además las instancias `CMT-n101-k10` y `CMTd-n101-k11`, formadas por clientes agrupados por zonas. Esta última (`CMTd-n101-k11`) la obtenemos después de incluir en `CMT-n101-k10` los tiempos de entrega y las longitudes de rutas máximas.

Los algoritmos estudiados en esta sección se comparan en términos de la media de la longitud total del recorrido de las soluciones encontradas, el número medio de evaluaciones requeridas para obtener dichas soluciones y la tasa de éxito (un tanto por ciento de las ejecuciones con éxito), en 30 ejecuciones independientes. Por tanto, estamos comparando los algoritmos según su precisión, eficiencia y eficacia. Realizamos tests estadísticos con objeto de comprobar la existencia de confianza estadística en la comparación de los resultados (ver Capítulo 4). Los parámetros utilizados para todos los algoritmos de este trabajo aparecen en la Tabla 13.1, con alguna excepción mencionada en cada caso. Los valores presentados en todas las tablas se corresponden con los valores medios encontrados, junto con su desviación estándar. Los mejores valores están resaltados en **negrita**.

13.3.1. Algoritmo Genético Celular vs. Generacional

En esta sección compararemos el comportamiento de nuestro GA celular, JCell2o1i, con uno generacional panmíctico (genGA) para ilustrar nuestra decisión de utilizar un modelo celular estructurado para resolver el CVRP. El mismo algoritmo se ejecuta en ambos casos diferenciándose únicamente en la estructura del vecindario del cGA. Como se puede observar, estructurar la población ha resultado ser muy útil para mejorar el algoritmo en muchos dominios, ya que la diversidad de la población se mantiene mejor con respecto a la población no estructurada. Los resultados de esta comparación se muestran en la Tabla 13.2. Como vemos, el genGA no consigue encontrar el óptimo para la mitad de las instancias probadas (`CMT-n76-k10`, `CMT-n101-k8`, `CMTd-n51-k6`, y `CMTd-n101-k9`), mientras que JCell2o1i siempre lo encuentra, con una mayor tasa de éxito y menor esfuerzo.

Tabla 13.2: Algoritmos genéticos celulares y generacionales.

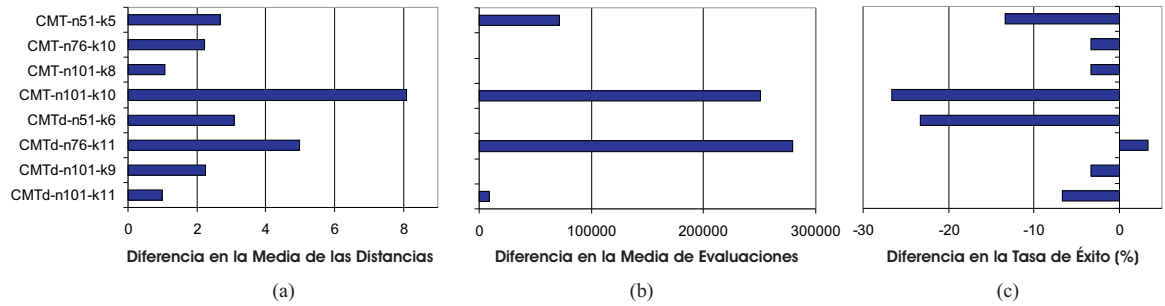
Instancia	**GA Generacional**			**JCell2o1i**		
	Sol. Media	Media de Evals.	éxito (%)	Sol. Media	Media de Evals.	Éxito (%)
`CMT-n51-k5`	527.88	98158.34	40.00	**525.19**	**26374.31**	**53.33**
	±3.70	*±23550.01*		*±1.52*	*±6355.15*	
`CMT-n76-k10`	845.00	—	0.00	**842.77**	**72102.00**	**3.33**
	±6.22	—		*±4.58*	*±0.00*	
`CMT-n101-k8`	833.50	—	0.00	**832.43**	**177248.00**	**3.33**
	±4.07	—		*±0.00*	*±0.00*	
`CMT-n101-k10`	828.98	321194.74	63.33	**820.89**	**70637.41**	**90.00**
	±14.65	*±122200.58*		*±4.88*	*±13104.67*	
`CMTd-n51-k6`	561.37	—	0.00	**558.28**	**27969.15**	**23.33**
	±3.03	—		*±2.05*	*±6050.44*	
`CMTd-n76-k11`	923.47	344500.00	**10.00**	**918.50**	**64964.50**	6.67
	±9.43	*±62044.26*		*±7.42*	*±25595.15*	
`CMTd-n101-k9`	879.20	—	0.00	**876.94**	**91882.00**	**3.33**
	±5.78	—		*±5.89*	*±0.00*	
`CMTd-n101-k11`	868.34	342742.86	63.33	**867.35**	**333926.32**	**70.00**
	±4.50	*±125050.93*		*±2.04*	*±125839.62*	

En la Figura 13.6 comparamos gráficamente los dos algoritmos dibujando las diferencias de los valores obtenidos (genGA menos cGA) en términos de precisión –longitud media de la solución– (Figura 13.6a), eficiencia –número medio de evaluaciones– (Figura 13.6b) y eficacia –tasa de éxito– (Figura 13.6c).

Como se puede observar, JCell2o1i siempre mejora al GA generacional para los tres parámetros comparados ($8 \times 3 = 24$ pruebas), excepto para la tasa de éxito en `CMTd-n76-k11`. El histograma de la Figura 13.6b está incompleto porque el genGA no encuentra solución para algunas instancias (véase la segunda columna de la Tabla 13.2). Tras aplicar un test estadístico sobre los resultados del número de evaluaciones realizadas y de la distancia de las soluciones, concluimos que siempre hay confidencia estadística (p-valores ≥ 0.05) para estos argumentos entre los dos algoritmos.

13.3.2. La Importancia del Operador de Mutación

Una vez que ya hemos mostrado claramente la gran superioridad del JCell respecto a un GA generacional, analizaremos a fondo el algoritmo. Por tanto, en esta sección estudiaremos el comportamiento de JCell utilizando cada uno de los tres operadores de mutación propuestos por separado, y compararemos los resultados con los de JCell2o1i, que utiliza una combinación de las tres mutaciones. En la Tabla 13.3 podemos observar los resultados obtenidos por los cuatro algoritmos (junto con sus desvia-



(a) (b) (c)

Figura 13.6: Comparación entre el GA generacional y JCell2o1i (valores de genGA menos JCell2o1i).

Tabla 13.3: Análisis de la mutación.

Inst.	**JCell2o1i$_{inv}$** Sol. Media	Media Evals.	%	**JCell2o1i$_{ins}$** Sol. Media	Media Evals.	%	**JCell2o1i$_{sw}$** Sol. Media	Media Evals.	%	**JCell2o1i** Sol. Media	Media Evals.	%
CMT-n51-k5	527.38 *±3.48*	1.52E5 *±8.23E4*	57	527.02 *±3.32*	2.09E5 *±6.53E5*	60	525.68 *±2.48*	1.48E5 *±7.59E4*	**80**	**525.19** *±1.52*	**2.64E4** *±6.36E3*	53
CMT-n76-k10	844.79 *±4.75*	—— ——	0	843.13 *±5.48*	6.19E5 *±0.00*	**3**	844.81 *±5.71*	—— ——	0	**842.77** *±4.58*	**7.21E4** *±0.00*	**3**
CMT-n101-k8	835.99 *±4.54*	—— ——	0	832.98 *±4.20*	5.25E5 *±0.00*	**3**	835.25 *±5.71*	—— ——	0	**832.43** *±0.00*	**1.77E5** *±0.00*	**3**
CMT-n101-k10	824.91 *±9.15*	3.05E6 *±1.30E5*	73	**820.27** *±3.60*	2.71E5 *±7.49E4*	**90**	826.98 *±9.28*	3.48E5 *±1.56E5*	57	820.89 *±4.88*	**7.06E4** *±1.31E4*	**90**
CMTd-n51-k6	558.86 *±2.62*	9.85E4 *±2.15E4*	17	558.61 *±2.37*	2.60E5 *±1.88E5*	20	559.89 *±3.06*	1.04E5 *±3.49E4*	10	**558.28** *±2.05*	**2.80E4** *±6.05E3*	**23**
CMTd-n76-k11	921.57 *±10.21*	4.01E5 *±1.90E5*	**10**	**917.71** *±7.42*	5.06E5 *±5.51E4*	7	918.76 *±8.40*	3.83E5 *±1.03E4*	**10**	918.50 *±7.42*	**6.50E4** *±2.56E4*	7
CMTd-n101-k9	876.08 *±6.51*	7.59E5 *±0.00*	3	**873.26** *±5.77*	5.55E5 *±1.54E5*	**13**	876.15 *±6.74*	6.03E5 *±4.12E5*	7	876.94 *±5.89*	**9.19E4** *±0.00*	3
CMTd-n101-k11	867.50 *±3.68*	3.50E5 *±1.31E5*	**90**	867.83 *±3.46*	3.86E5 *±1.38E5*	77	867.40 *±4.24*	4.51E5 *±1.73E5*	83	**867.35** *±2.04*	**3.34E5** *±1.26E5*	70

ciones estándar). Estos algoritmos difieren unos de otros únicamente en el método de mutación aplicado: inversión (JCell2o1i$_{inv}$), inserción (JCell2o1i$_{ins}$), intercambio (JCell2o1i$_{sw}$), o los tres (JCell2o1i) —para más detalles sobre los operadores de mutación consulte la Sección 13.2.3.

Una observación interesante es el hecho de que el uso de un único operador de mutación no parece aportar un claro ganador, es decir, no podemos concluir una mejora total general de ninguna de las tres variantes básicas celulares. Por ejemplo, usando la mutación por *inversión* (JCell2o1i$_{inv}$) obtenemos soluciones muy precisas para la mayoría de las instancias, aunque es menos precisa para algunas otras, como `CMT-n51-k5`, `CMT-n101-k8`, y `CMTd-n76-k11`. El mismo comportamiento obtenemos en términos del número medio de evaluaciones realizadas y de la tasa de éxito; por tanto está claro que el mejor algoritmo depende de la instancia a resolver cuando utilizamos un único operador de mutación. Por eso, inspirados en el trabajo de Chellapilla [55], en el que el autor logra mejorar el rendimiento de su algoritmo (programación evolutiva) mediante la combinación de dos mutaciones distintas, nosotros hemos decidido desarrollar una nueva mutación compuesta por las tres anteriormente propuestas (llamada mutación *combinada*) para explotar el comportamiento de la mejor de ellas en cada instancia. Como podemos comprobar en la Tabla 13.3, esta idea funciona: JCell2o1i es el mejor de los cuatro métodos comparados. Es el mejor algoritmo para todas las instancias probadas en lo que a número medio de evaluaciones realizadas se refiere. Es más, en términos de precisión y tasa de éxito, JCell2o1i es el algoritmo que encuentra el mejor valor para un gran número de instancias (véanse los valores en **negrita** para las tres métricas).

En la Figura 13.7 mostramos una evaluación gráfica de los cuatro algoritmos en términos de precisión, eficiencia y eficacia. JCell2o1i es, en general, más preciso que los otros tres algoritmos, aunque las diferencias son muy pequeñas (Figura 13.7a). En términos del número medio de evaluaciones realizadas (eficiencia), el algoritmo que utiliza la mutación *combinada* aventaja claramente a los otros tres (Figura 13.7b), mientras que los valores obtenidos en el caso de la tasa de éxito son un poco peores, pero sólo para unas pocas instancias (Figura 13.7c). Después de aplicar el análisis de ANOVA a los resultados obtenidos de los algoritmos, no se encontraron diferencias estadísticas, en general, en lo que respecta a la precisión, pero el nuevo método de mutación combinada es mejor que los otros tres con confidencia estadística para las ocho instancias probadas si prestamos atención al número de evaluaciones realizadas (esfuerzo). Esto significa que nuestro algoritmo es competitivo en cuanto a precisión, pero mucho mejor en cuanto a eficiencia.

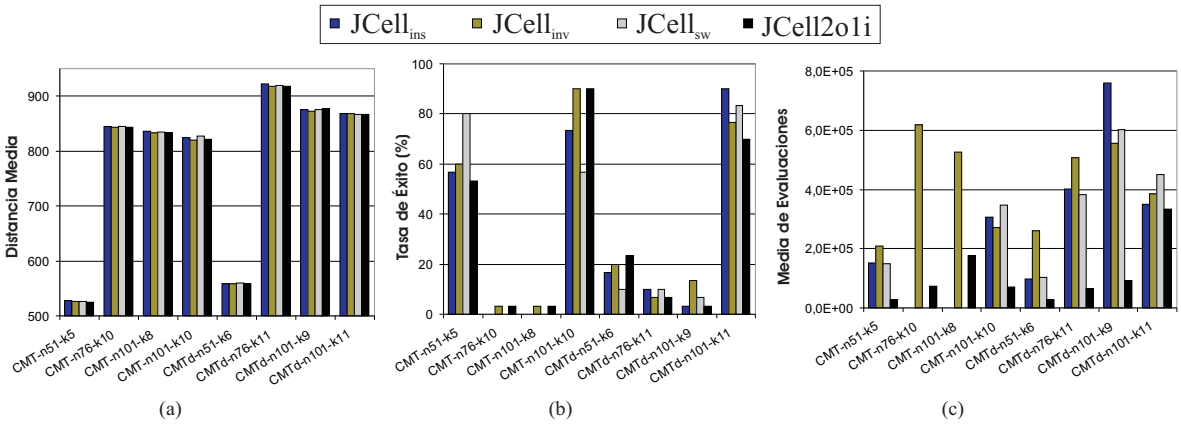


Figura 13.7: Comparación del comportamiento de un cGA usando las tres mutaciones diferentes independientemente y juntas en términos de (a) precisión, (b) esfuerzo, y (c) eficacia.

13.3.3. Ajustando la Etapa de Búsqueda Local

Se ha mostrado que la mutación es un componente muy importante para reducir el esfuerzo del algoritmo, pero la búsqueda local también es un operador muy influyente, y en esta sección analizaremos en qué medida afecta. En la etapa de búsqueda local, nuestra propuesta incluye 2-Opt y 1-Intercambio, pero surge la pregunta natural de si se podrían haber utilizado algunas variantes (más sofisticadas) en su lugar. Para contestar a esta pregunta probaremos en esta sección el comportamiento del JCell (sin búsqueda local) con algunas diferentes posibilidades de hibridarlo. Todas las opciones híbridas estudiadas se basan en 2-Opt y λ-Intercambio (dos métodos de búsqueda local muy conocidos para VRP): JCell con 2-Opt (JCell2o), con 1-Intercambio (JCell1i) y dos combinaciones distintas de 2-Opt y λ-Intercambio; aquellas con $\lambda = 1$ (JCell2o1i) y $\lambda = 2$ (JCell2o2i). Mostramos los resultados en la tablas 13.4 y 13.5 (ha sido necesario dividir los resultados en dos tablas por motivos de formato del documento).

A partir de las tablas 13.4 y 13.5 podemos concluir que es posible mejorar el rendimiento de un cGA con la mutación *combinada* para el problema VRP simplemente incluyendo una etapa de búsqueda local. Se puede observar que JCell2o obtiene mejores resultados que JCell (sin búsqueda local) en todas las instancias estudiadas (algoritmos en las columnas 1 y 2 en la Tabla 13.4), pero la mejora es notable cuando el método de λ-Intercambio se considera con o sin 2-Opt. La importancia de este operador de búsqueda local es notable, ya que el algoritmo no es capaz de encontrar la mejor solución al problema para ninguna instancia probada cuando no usamos λ-Intercambio. Aplicando el análisis estadístico a las soluciones obtenidas por los algoritmos, podemos concluir que los tres algoritmos que utilizan λ-Intercambio son mejores que los otros con confianza estadística en términos de precisión. Las diferencias entre estos tres algoritmos en el número medio de evaluaciones realizadas también tiene (con menor connotación) significancia estadística.

Con respecto a los dos algoritmos con los mejores resultados generales, aquellos que usan 2-Opt y λ-Intercambio (Tabla 13.5), JCell2o1i siempre obtiene mejores valores que JCell2o2i (véanse los valores en **negrita**) en términos del número medio de evaluaciones realizadas (con significancia estadística). Por tanto, aplicar 1-Intercambio es más rápido que aplicar 2-Intercambio (como se podía esperar ya que es una operación más trivial) y obtenemos mayor precisión y eficiencia (menos recursos computacionales).

Tabla 13.4: Análisis de los efectos del uso de la búsqueda local en JCell con mutación combinada (1).

	JCell (no LS)			**JCell2o**			**JCell1i**		
Inst.	Dist. Media	Media Evals.	Éxito (%)	Dist. Media	Media Evals.	Éxito (%)	Dist. Media	Media Evals.	Éxito (%)
CMT-n51-k5	576.1 ±14.4	— —	0	551.6 ±9.4	— —	0	529.9 ±6.7	$1.0E5$ ±3.7E4	50
CMT-n76-k10	956.4 ±20.5	— —	0	901.8 ±15.8	— —	0	851.5 ±6.3	— —	0
CMT-n101-k8	994.4 ±29.9	— —	0	867.2 ±14.5	— —	0	840.6 ±6.5	— —	0
CMT-n101-k10	1053.6 ±43.5	— —	0	949.6 ±29.0	— —	0	822.8 ±5.9	$2.3E5$ ±4.2eE5	10
CMTd-n51-k6	611.5 ±15.2	— —	0	571.9 ±13.4	— —	0	561.7 ±4.0	$6.9E4$ ±5.9E3	3
CMTd-n76-k11	1099.1 ±26.1	— —	0	1002.6 ±21.7	— —	0	924.0 ±8.2	$2.0E5$ ±0.0	3
CMTd-n101-k10	1117.5 ±34.8	— —	0	936.5 ±15.1	— —	0	882.2 ±10.4	$3.8E5$ ±0.0	13
CMTd-n101-k11	1110.1 ±58.8	— —	0	1002.8 ±40.4	— —	0	869.5 ±3.5	**1.8E5** ±6.7E4	13

Tabla 13.5: Análisis de los efectos del uso de la búsqueda local en JCell con mutación combinada (y 2).

	JCell2o1i			**JCell2o2i**		
Inst.	Dist. Media	Media Evals.	Éxito (%)	Dist. Media	Media Evals.	Éxito (%)
CMT-n51-k5	**525.2** ±1.5	**2.6E4** ±6.4E3	53	526.1 ±2.9	$1.6E5$ ±6.8E4	**77**
CMT-n76-k10	**842.8** ±4.6	**7.2E4** ±0.0	**3**	844.7 ±5.3	$7.5E5$ ±0.0	**3**
CMT-n101-k8	832.4 ±0.0	**1.8E4** ±0.0	**3**	**831.9** ±7.2	$8.2E5$ ±0.0	**3**
CMT-n101-k10	**820.9** ±4.9	**7.1E4** ±1.3E4	**90**	823.3 ±7.6	$1.2E5$ ±2.7e4	17
CMTd-n51-k6	**558.3** ±2.1	**2.8E4** ±6.1E3	**23**	558.6 ±2.5	$3.7E5$ ±1.8E5	13
CMTd-n76-k11	918.50 ±7.4	**6.5E4** ±2.6E4	7	**917.7** ±9.1	$7.1E5$ ±3.5E5	**13**
CMTd-n101-k10	876.94 ±5.9	**9.2E4** ±0.0	3	**873.7** ±4.9	$2.87E5$ ±1.2E5	**80**
CMTd-n101-k11	**867.4** ±2.0	$3.3E5$ ±1.3E5	70	867.9 ±3.9	$3.5E5$ ±1.4E5	**83**

Resumimos en la Figura 13.8 los resultados de los algoritmos comparados. JCell y JCell2o no aparecen en las Figuras 13.8b y 13.8c porque ninguno de estos dos algoritmos son capaces de encontrar la mejor solución conocida para ninguna de las instancias probadas. En términos de la distancia media de las soluciones encontradas (Figura 13.8a), se puede observar que los tres algoritmos que utilizan el método de λ-Intercambio tienen resultados similares, mejorando al resto en todos los casos. JCell2o1i es el algoritmo que necesita el menor número de evaluaciones para alcanzar la solución en casi todos los casos (Figura 13.8b). En términos de la tasa de éxito (Figura 13.8c), JCell1i es, en general, el peor algoritmo (entre los tres que utilizan λ-Intercambio) para todas las instancias probadas.

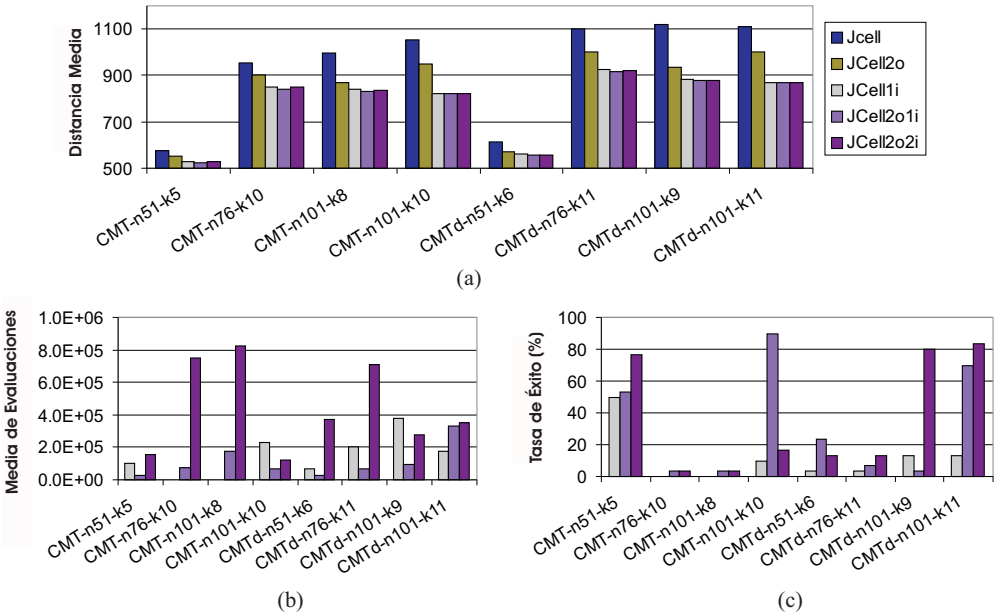


Figura 13.8: Comparación entre el comportamiento de un cGA sin búsqueda local y cuatro cGAs usando diferentes técnicas de búsqueda local en términos de (a) precisión, (b) esfuerzo, y (c) eficacia.

Para resumir con un ejemplo los estudios realizados en esta sección, dibujamos en la Figura 13.9 la evolución de la mejor solución (Figura 13.9a) y de la solución media (Figura 13.9b) que se encuentran durante una ejecución con los cinco algoritmos utilizados en esta sección para `CMT-n51-k5`. Todos los algoritmos implementan la mutación *combinada*, que se compone de los tres métodos estudiados en la Sección 13.2.3: inserción, inversión e intercambio. En las gráficas 13.9a y 13.9b podemos observar un comportamiento similar, en general, para todos los algoritmos: la población evoluciona rápidamente a

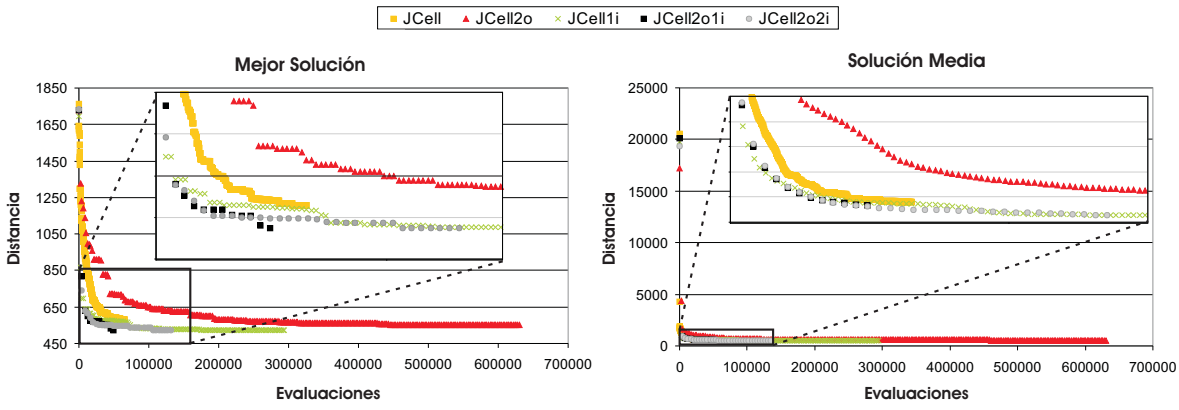


Figura 13.9: Evolución de las mejores (a) y medias (b) soluciones para `CMT-n51-k5` con algunos algoritmos que sólo se diferencian en la búsqueda local aplicada.

soluciones cercanas a la mejor conocida (valor 524.61) en todos los casos. Si nos fijamos en las gráficas podemos notar más claramente que tanto JCell como JCell2o convergen más despacio que los otros, y que finalmente se atascan en un óptimo local. JCell2o mantiene la diversidad durante más tiempo que los otros algoritmos. Aunque los algoritmos con λ-Intercambio convergen más rápido que JCell y que JCell2o, y además son capaces de encontrar el valor óptimo, escapando de óptimos locales gracias al método de búsqueda local λ-Intercambio.

13.4. Resolviendo CVRP con JCell2o1i

En esta sección describimos los resultados de los experimentos que hemos realizado para probar nuestro algoritmo sobre CVRP. JCell2o1i se ha implementado en Java y lo hemos probado sobre un PC a 2.6 GHz trabajando bajo Linux con un grupo de pruebas muy grande, compuesto por un conjunto de problemas muy extenso escogido de la literatura: (i) Augerat *et al.* (conjuntos A, B y P) [32], (ii) Van Breedam [279], (iii) Christofides y Eilon [57], (iv) Christofides, Mingozzi y Toth (CMT) [58], (v) Fisher [99], (vi) Golden *et al.* [125], (vii) Taillard [266], y (viii) un conjunto generado desde instancias del TSP [237]. Todas estas baterías de problemas, junto con las mejores soluciones encontradas para las instancias que las componen, están públicamente disponibles en [84].

Debido a la heterogeneidad de nuestro gran conjunto de pruebas, encontramos muchas características distintas en las instancias estudiadas, lo que representará un duro test para JCell2o1i en este problema. El conjunto de problemas estudiado en este capítulo es mucho más grande que el utilizado en los estudios habituales en la literatura para un único tipo de VRP. Los parámetros que utiliza JCell2o1i en todas las pruebas se mostraron previamente en la Tabla 13.1.

En las secciones 13.4.1 a 13.4.8 analizaremos en detalle los resultados obtenidos con JCell2o1i para las 160 instancias que componen nuestro conjunto de pruebas. Los resultados numéricos de los propios algoritmos (rutas óptimas, costes, etc.) se muestran en el Apéndice B, donde las cifras en las tablas se calculan tras realizar 100 ejecuciones independientes (para obtener confianza estadística), excepto para la Tabla B.17, donde sólo se hacen 30 ejecuciones debido a la dificultad computacional de los problemas. Las mejores soluciones encontradas aquí por primera vez se muestran también en el Apéndice B.

Durante todo este capítulo utilizamos la misma nomenclatura para todas las instancias. En esta nomenclatura, los nombres de las instancias se componen de un identificador del conjunto al que pertenece la instancia, seguido de una 'n' y el número de nodos (clientes a servir más el almacén) y una 'k' con el número de vehículos utilizados en la mejor solución (por ejemplo, `bench-nXX-kXX`).

En esta sección las métricas para la presentación del algoritmo son (véanse la tablas de resultados en el Apéndice B) el valor de la mejor solución encontrada en cada instancia, el número medio de evaluaciones realizadas hasta que se encuentra la mejor solución, la tasa de éxito, el valor medio de las soluciones encontradas en todas las ejecuciones independientes, la desviación entre nuestra mejor solución y la mejor hasta ahora conocida para esa instancia (en tanto por ciento), y la mejor solución previamente conocida para cada instancia. La desviación entre nuestra mejor solución (sol) y la mejor solución conocida hasta el momento ($best$) se calcula con la Ecuación 13.8.

$$\Delta(sol) = \left[\frac{best - sol}{best}\right] \cdot 100 \ . \tag{13.8}$$

En las secciones 13.4.1, 13.4.2, 13.4.3, 13.4.5, y13.4.8, las distancias entre los clientes se han redondeado al valor entero más cercano, siguiendo la convención de TSPLIB [237]. En las demás secciones no se ha aplicado redondeo. Discutiremos cada conjunto de pruebas en una sección separada.

13.4.1. Problemas de Augerat et al.

Esta batería de problemas, propuesto en 1995, se compone de tres conjuntos de problemas (conjuntos A, B y P). Todas las instancias de cada conjunto tienen distintas características, como la distribución de la posición de los clientes. Se ha comprobado que las mejores soluciones conocidas son los óptimos para cada instancia de este conjunto de problemas [191]. Estudiaremos el comportamiento de JCell2o1i al resolver los tres subconjuntos que componen esta batería de problemas. La desviación de las soluciones encontradas no se han dibujado en esta sección porque JCell2o1i encuentra las mejores soluciones conocidas en todas las instancias de la batería de problemas (excepto para `B-n68-k9`).

Conjunto A

Este conjunto está constituido por instancias donde tanto las posiciones de los clientes como las demandas se generan aleatoriamente mediante una distribución uniforme. El tamaño de las instancias está en el rango de 31 a 79 clientes. Podemos ver en la Tabla B.10 que JCell2o1i resuelve óptimamente todas las instancias. En la Figura 13.10 dibujamos el porcentaje de ejecuciones en el que el algoritmo es capaz de encontrar el óptimo para cada instancia (tasa de éxito). El lector notará que, en general, para el algoritmo es más difícil encontrar la solución óptima conforme el número de clientes crece, ya que la tasa disminuye al aumentar el tamaño de las instancias. Efectivamente, la tasa de éxito está por debajo del 8 % para instancias que tienen más de 59 clientes, aunque también es baja para instancias pequeñas como `A-n39-k5` y `A-n45-k6`.

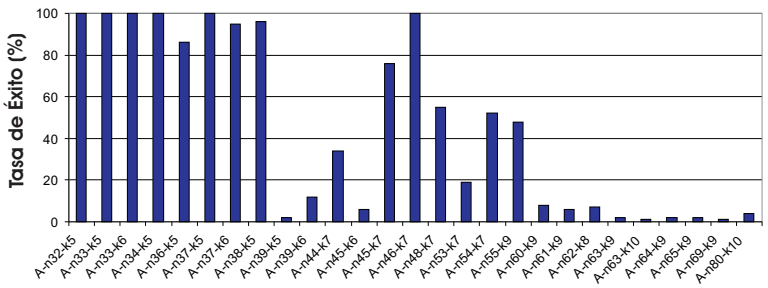


Figura 13.10: Tasas de éxito para el conjunto de problemas de Augerat et al., conjunto A.

Conjunto B

Las instancias de este conjunto se caracterizan principalmente por estar los clientes agrupados en zonas localizadas. Este conjunto de instancias no parece ser realmente un reto para nuestro algoritmo ya que es posible encontrar el óptimo para todas ellas (Tabla B.11). Sólo hay una excepción (la instancia B-n68-k9) pero la desviación entre nuestra mejor solución y la mejor solución conocida es realmente baja (0.08 %). En la Figura 13.11 mostramos la tasa de éxito obtenida para cada instancia. Como puede observarse, dicha tasa de éxito es menor del 5 % sólo para algunas instancias (`B-n50-k8`, `B-n63-k10`, y `B-n66-k9`).

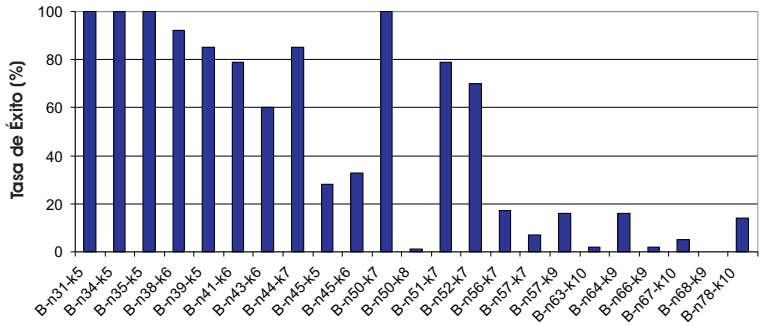


Figura 13.11: Tasas de éxito para el conjunto de problemas de Augerat et al., conjunto B.

Conjunto P

Las instancias de la clase 'P' son versiones modificadas de otras instancias tomadas de la literatura. En la Tabla B.12 resumimos nuestros resultados. El lector puede observar que nuestro algoritmo también puede encontrar todas las soluciones óptimas para este conjunto. La tasa de éxito se muestra en la Figura 13.12, y aparece la misma tendencia que en los dos casos anteriores: para las instancias más grandes (más de 60 clientes) se obtienen tasas de éxito bajas.

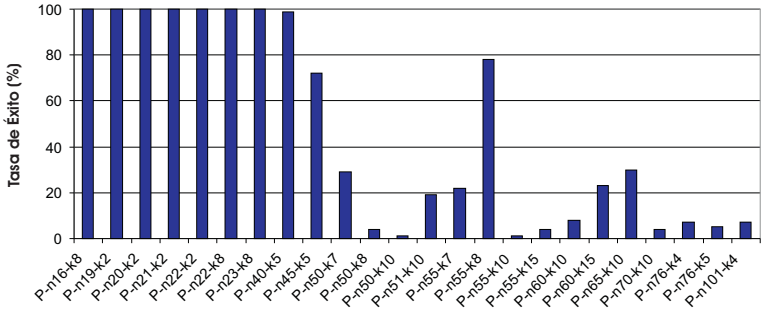


Figura 13.12: Tasas de éxito conjunto de problemas de Augerat et al., conjunto P.

Para terminar esta sección sólo puntualizaremos que el algoritmo es por sí mismo competitivo con muchos otros algoritmos en los tres conjuntos de instancias, lo que supone una propiedad interesante.

13.4.2. Problemas de Van Breedam

Van Breedam propuso en su tesis [279] un gran conjunto de 420 instancias con muchas características diversas, como distintas distribuciones de clientes (solos, agrupados, cónicos,...) y de almacén (central, interior, exterior) en el espacio, con ventanas de tiempo, con recogida y entrega o con demandas heterogéneas. Van Breedam también propuso un conjunto reducido de instancias representativas del inicial (compuesto por 15 problemas) y lo resolvió con muchas heurísticas distintas [279] (existen trabajos más recientes que utilizan este conjunto, pero no aportan ninguna mejor solución [280, 281]). Este reducido conjunto de instancias es el que estudiaremos en este capítulo.

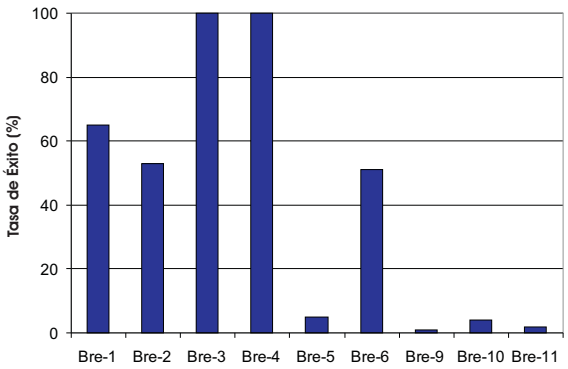


Figura 13.13: Tasas de éxito del conjunto de problemas de Van Breedam.

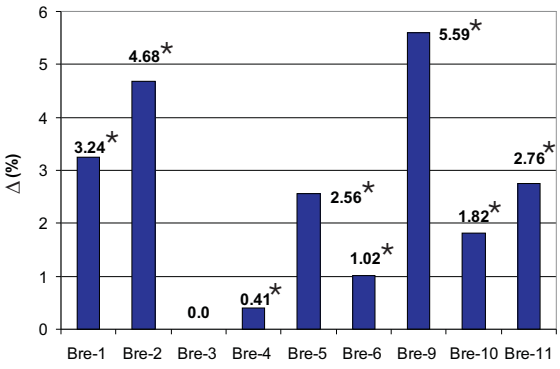


Figura 13.14: Porcentajes de desviación del conjunto de problemas de Van Breedam.

Todas las instancias de esta batería de problemas se componen del mismo número de clientes ($n = 100$), y la demanda total de éstos es siempre la misma, 1000 unidades. Si adoptamos la nomenclatura utilizada en este capítulo, las instancias que utilizan la misma capacidad de vehículos tendrán el mismo nombre; para evitar que se repitan los nombres para las distintas instancias, utilizaremos una nomenclatura especial para este conjunto, numerando los problemas de `Bre-1` a `Bre-15`. Los problemas de `Bre-1` a `Bre-6` sólo tienen restricciones en la capacidad de los vehículos. La demanda en cada parada es de 10 unidades. La capacidad del vehículo es de 100 unidades para `Bre-1` y `Bre-2`, 50 unidades para `Bre-3` y `Bre-4`, y 200 unidades para `Bre-5` y `Bre-6`. Los problemas `Bre-7` y `Bre-8` no se estudian en este trabajo porque incluyen entregas y recogidas. Una característica específica de este conjunto de instancias es el uso de la demanda homogénea en las paradas, lo que significa que todos los clientes de una instancia requieren la misma cantidad de artículos. Las excepciones son los problemas que van del `Bre-9` al `Bre-11`, para los que las demandas de los clientes son heterogéneas. Los restantes cuatro problemas, del `Bre-12` al `Bre-15`, no se estudian en este trabajo porque incluyen ventanas de tiempo (fuera de nuestro ámbito).

Nuestra primera conclusión es que JCell2o1i supera las mejores soluciones conocidas hasta el momento para ocho de las nueve instancias. Es más, la solución media encontrada en todas las ejecuciones es mejor que la solución previamente conocida en estos ocho problemas (véase la Tabla B.13), esto es una indicación de la gran precisión y estabilidad de nuestro algoritmo. Como podemos ver en la Figura 13.13, el algoritmo encuentra la nueva mejor solución en un alto porcentaje de ejecuciones en algunas instancias (`Bre-1`, `Bre-2`, `Bre-4`, Y `Bre-6`), mientras que en otras la nueva mejor solución se encuentra sólo en algunas ejecuciones (`Bre-5`, `Bre-9`, `Bre-10`, y `Bre-11`). En la Figura 13.14 mostramos las desviaciones de las soluciones encontradas (el símbolo $\star$ marca las instancias mejoradas). Las nuevas mejores soluciones encontradas para este conjunto de instancias se muestran en las tablas B.1 a B.8 en el Apéndice B.

No hemos encontrado patrones en común en las distintas instancias (ni en la distribución de clientes ni en la posición del almacén) que nos permitan predecir la precisión del algoritmo cuando las resolvemos. Hemos notado que las 4 instancias para las que se obtiene una menor tasa de éxito (`Bre-5`, `Bre-9`, `Bre-10`, y `Bre-11`) tienen características en común, como la localización no centralizada del almacén y la presencia de un único cliente situado alejado del resto, pero estas dos características se pueden encontrar también en las instancias `Bre-2` y `Bre-6`. De todas maneras, un aspecto que tienen en común las instancias

de la `Bre-9` a la `Bre-11` (que no está presente en otras instancias de este conjunto) son las demandas heterogéneas en las paradas. Efectivamente, podemos concluir que el uso de demandas heterogéneas en este conjunto reduce el número de ejecuciones en las que el algoritmo obtiene la mejor solución encontrada (tasa de éxito). Finalmente, en el caso de `Bre-3` (agrupaciones de clientes distribuidas uniformemente de forma circular y almacenes no centrados), la mejor solución conocida no pudo ser mejorada, pero se encontró en todas las ejecuciones.

13.4.3. Problemas de Christofides y Eilon

Las instancias que componen esta batería de problemas van desde problemas sencillos de 21 clientes a otros más difíciles de hasta 100 clientes. Podemos distinguir dos tipos de instancias en el conjunto de problemas. En las instancias más pequeñas (hasta 32 clientes), el almacén y todos los clientes (excepto uno) están en una pequeña región del plano y existe un único cliente alejado de dicha región (como en el caso de algunas instancias de los problemas de Van Breedam). En el resto de instancias de este conjunto, los clientes se distribuyen aleatoriamente en el plano y el almacén está en el centro o cerca de él.

Para comparar nuestros resultados con los que encontramos en la literatura, la distancia entre clientes se ha redondeado al valor entero más cercano, tal y como ya hicimos antes con los tres conjuntos de instancias de Augerat et al. en la Sección 13.4.1. Mostramos en la Figura 13.15 el porcentaje de ejecuciones en las que se encuentra el óptimo. Se puede ver en este esquema cómo el algoritmo encuentra algunas dificultades al resolver las instancias más grandes (más de 50 clientes y los clientes distribuidos por todo el área del problema), para los que la tasa de éxito va desde 1 % al 8 %. En el caso de las instancias más pequeñas (32 clientes o menos), la solución se encuentra en el 100 % de las ejecuciones, excepto para E-n31-k7 (67 %). Las desviaciones de las soluciones encontradas con respecto a las mejores conocidas (Δ) no se dibujan porque JCell2o1i es capaz de encontrar las soluciones óptimas ($\Delta = 0.0$) para todas las instancias de esta batería de problemas (véase la Tabla B.14).

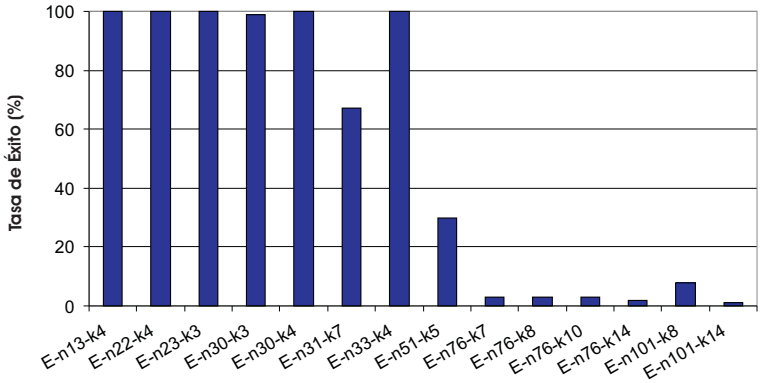


Figura 13.15: Tasas de éxito del conjunto de problemas de Christofides y Eilon.

13.4.4. Problemas de Christofides, Mingozzi y Toth

El conjunto de problemas que presentamos en esta sección se ha explorado a fondo por muchos investigadores. Se compone de diez instancias en las que se sitúan a los clientes de manera aleatoria en el plano, más cuatro instancias con clientes agrupados por zonas (CMT-n101-k10, CMT-n121-k7, CMTd-n101-k11, y CMTd-n121-k11). Entre los catorce problemas que componen este conjunto, hay siete problemas básicos de VRP que no incluyen restricciones en la longitud de las rutas (los llamados CMT-nXX-kXX). Las otras siete instancias (llamadas CMTd-nXX-kXX) usan la misma localización de los clientes que las siete anteriores, pero están sujetas a restricciones adicionales, como las distancias de las rutas limitadas y los tiempos de entrega en las visitas a cada cliente (tiempo necesario para descargar los artículos). Estas instancias son distintas a todos los conjuntos de problemas estudiados en este trabajo porque tienen asociados a los clientes tiempos de entrega. Estos tiempos de entrega son homogéneos para todos los clientes de una instancia (es necesario el mismo tiempo para descargar los artículos demandados en todos los clientes). Por otro lado, las demandas en las paradas son heterogéneas, así que distintos clientes pueden solicitar distintas cantidades de artículos.

Las cuatro instancias con clientes agrupados por zonas no tienen almacén central, mientras que en las otras el almacén está situado en el centro (con la excepción de los problemas CMT-n51-k5 y CMTd-n51-k6).

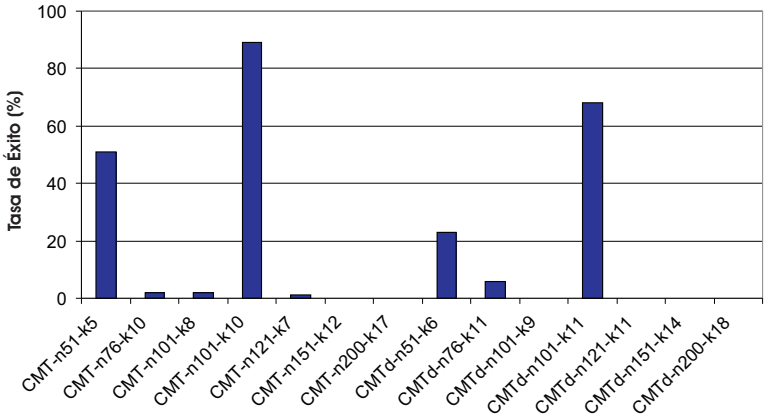


Figura 13.16: Tasas de éxito para el conjunto de problemas de Christofides, Mingozzi y Toth.

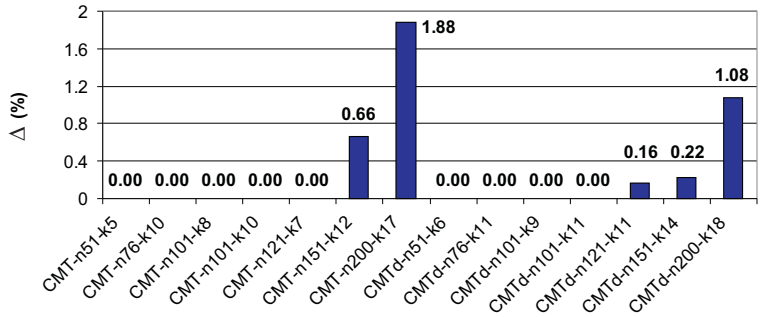


Figura 13.17: Tasas de desviación para el conjunto de problemas Christofides, Mingozzi y Toth.

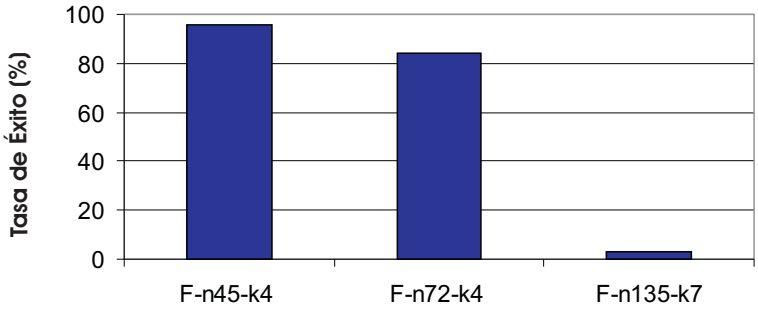


Figura 13.18: Tasas de éxito del conjunto de problemas de Fisher.

Como puede observar el lector en la Tabla B.15, estas instancias son, en general, más difíciles que las anteriormente estudiadas ya que, por primera vez en nuestro estudio, JCell2o1i no puede encontrar la solución óptima para alguna de estas instancias (véase Figura 13.16). Específicamente, no podemos encontrar las mejores soluciones conocidas en nuestro algoritmo para seis de las instancias (las más grandes). Sin embargo, se puede observar en la Figura 13.17 que la diferencias (Δ) entre nuestra mejor solución y la mejor solución conocida es siempre menor del 2 %. La mejor solución conocida la hemos encontrado en todas las instancias de clientes agrupados, excepto para CMTd-n121-k11 ($\Delta = 0.16$). Nótese que las dos instancias no agrupadas con un almacén no centrado se resolvieron un 51 % (CMT-n51-k5) y 23 % (CMTd-n51-k6) de las ejecuciones. Por tanto, la mejor solución conocida se encontró para todas aquellas instancias que no tienen almacén en el centro, con la excepción de CMTd-n121-k11.

13.4.5. Problemas de Fisher

Este conjunto de problemas se compone sólo de tres instancias, que se toman de aplicaciones reales de VRP. Los problemas `F-n45-k4` y `F-n135-k7` representan un día en las entregas de tiendas de comestibles de la compañía *National Grocers Limited*. El otro problema (`F-n72-k4`) está relacionado con la entrega de neumáticos, baterías y accesorios en estaciones de servicio, y se basa en datos obtenidos de Exxon (EE.UU.). El almacén no está en el centro en ninguna de las tres instancias. Al pertenecer este conjunto de problemas a TSPLIB, las distancias entre los clientes se redondean al valor entero más cercano. Una vez más, para todas las instancias se ha alcanzado el óptimo con nuestro algoritmo (véase la Tabla B.16) con una alta tasa de éxito en dos de las tres instancias (tasa de éxito del 3 % para F-n135-k7). En la Figura 13.18 mostramos las tasas de éxito obtenidas. Esta figura muestra claramente cómo crece la dificultad para encontrar el óptimo conforme aumenta el tamaño de las instancias. Una vez más, las desviaciones (Δ) no se han dibujado porque son 0.00 para las tres instancias.

13.4.6. Problemas de Golden, Wasil, Kelly y Chao

En esta sección vamos a tratar con un difícil conjunto de instancias aún más grande, que abarca entre 200 y 483 clientes. Las instancias que componen esta batería de problemas son teóricas, y los clientes están dispersos en el plano, formando figuras geométricas, como círculos concéntricos (8 instancias), cuadrados (4 instancias), rombos (4 instancias) y estrellas (4 instancias). El almacén está situado en el centro en todas las instancias excepto en las cuatro que tienen forma de rombo, en la que el almacén se sitúa en uno de los vértices. Sólo en las instancias llamadas `gold-nXXX-kXX` se consideran distancias de rutas máximas. En las otras instancias (`gol-nXXX-kXX`), las distancias de las rutas no tienen límite. No se consideran tiempos de entrega en este conjunto de problemas.

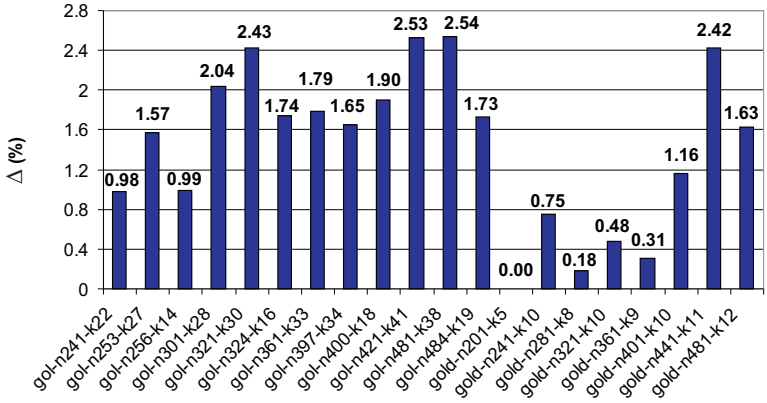


Figura 13.19: Tasas de desviación para el conjunto de problemas de Golden et al.

Los resultados se dan en la Tabla B.17 del Apéndice B. Como se puede observar, las instancias de esta batería de problemas son especialmente difíciles de resolver: sólo para la instancia `gold-n201-k5` encontramos la mejor solución conocida (con una tasa de éxito del 2 %). De todas maneras, la desviación entre el mejor valor conocido y el mejor encontrado es siempre muy baja (como en el caso de las secciones anteriores): por debajo del 2.60 % (véase la Figura 13.19).

También hemos notado que el tiempo de ejecución de nuestro algoritmo crece con el tamaño de las instancias. Esto se debe a la exhaustiva etapa de búsqueda local que hemos aplicado en cada generación a cada individuo. En esta etapa de optimización local, el algoritmo explora todos los vecindarios de un individuo dado obtenidos tras aplicar los métodos 2-Opt y 1-Intercambio. Por tanto, el número de soluciones a explorar crece exponencialmente con el número de clientes de la instancia.

En cualquier caso, la precisión de JCell2o1i no disminuye necesariamente con el incremento en el tamaño de la instancia, tal y como comprobamos en la Figura 13.19. Por ejemplo, la precisión al resolver `gol-n484-k19` es más alta (menor desviación con respecto a la mejor solución conocida) que en el caso de las instancias más pequeñas como `gol-n481-k38`, `gol-n421-k41`, `gol-n400-k18`, o incluso siendo el tamaño inferior a 100 clientes menos (`gol-n361-k33` y `gol-n324-k16`).

13.4.7. Problemas de Taillard

En esta sección presentamos un conjunto de problemas de VRP duros propuestos por Taillard. Se compone de 15 problemas no uniformes. Los clientes se sitúan en el plano, esparciéndose en varias agrupaciones, y tanto el número de agrupaciones como su densidad son bastante variables. Las cantidades que demandan los clientes son distribuidas exponencialmente (por lo que la demanda de un cliente podría ser igual la capacidad total de un vehículo).

Hay cuatro instancias de 75 clientes (9 ó 10 vehículos), otras cuatro de 100 clientes (10 u 11 vehículos) y cuatro más de 150 clientes (14 ó 15 vehículos). También tenemos un problema pseudo-real con 385 ciudades, que se generó de la siguiente manera: cada ciudad es la ciudad o pueblo más importante de la entidad política más pequeña (municipio) del *Cantón de Vaud*, en Suiza. El censo de los habitantes de cada municipio se tomó a principios de 1990. Consideramos una demanda de 1 unidad por cada 100 habitantes (pero al menos 1 de cada municipio) y la capacidad de los vehículos es de 65 unidades.

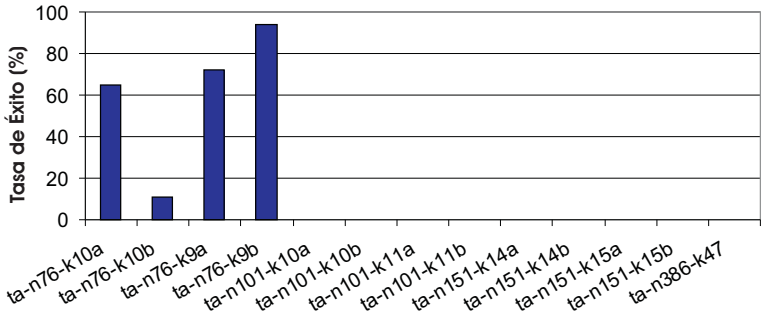


Figura 13.20: Tasas de éxito para el conjunto de problemas de Taillard.

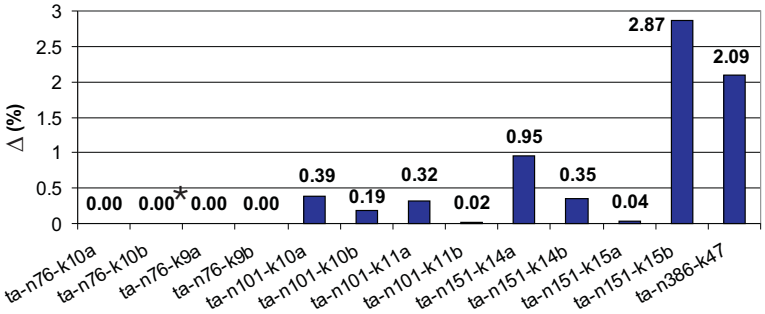


Figura 13.21: Tasas de desviación para el conjunto de problemas de Taillard ($\star$ marca la nueva mejor solución que ha encontrado JCell2o1i).

Como podemos ver en la Tabla B.18, JCell2o1i es capaz de obtener las mejores soluciones conocidas para las cuatro instancias pequeñas (las de 75 clientes) y también para `ta-n101-k11b` (véanse las tasas de éxito en la Figura 13.20). Además, para la instancia `ta-n76-k10b` nuestro algoritmo ha mejorado la mejor solución conocida hasta el momento en un 0.0015 % al compararla con la solución anterior (nueva solución = 1344.62); mostramos esta solución en la Tabla B.9 del Apéndice B. Observe que esta mejor solución está bastante cerca de la que existía previamente, pero representan soluciones muy distintas en términos de rutas resultantes. JCell2o1i ha encontrado la mejor solución previa para esta instancia en el 30 % de las ejecuciones realizadas. En las instancias que se encuentra la mejor solución pero no se mejora, la tasa de éxito está por encima del 60 %. Las desviaciones en las otras instancias, como en el caso de la sección anterior, son muy bajas (por debajo de 2.90) como podemos ver en la Figura 13.21, por lo que el algoritmo es muy estable.

13.4.8. Problemas de Instancias Tomadas del TSP

Esta batería de problemas está compuesta por pequeñas instancias (cuyo tamaño oscila entre 15 y 47 clientes) cuyos datos han sido reutilizados de las conocidas instancias del *Problema del viajante de comercio* (TSP). Como en el caso de algunas instancias del conjunto de problemas de Van Breedam, las

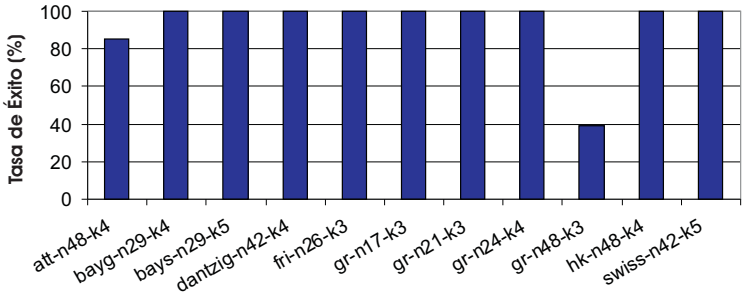


Figura 13.22: Tasas de éxito para el conjunto de problemas de instancias tomadas del TSP.

demandas de los clientes son homogéneas en estos problemas, es decir, todos los clientes demandan la misma cantidad de artículos. Una vez más, JCell2o1i alcanza el óptimo para todas las instancias (véase la Tabla B.19), y este óptimo se encuentra en un porcentaje muy alto de ejecuciones para cada una de ellas. De hecho, la solución óptima se encuentra en el 100 % de las ejecuciones para cada instancia, excepto para `att-n48-k4` (85 %) y `gr-n48-k3` (39 %), como se muestra en la Figura 13.22. Por tanto, la solución media encontrada en las 100 ejecuciones es muy próxima a la óptima, cuya tasa de éxito oscila desde el 0.00 % en los casos en que se obtienen tasas de éxito del 100 % al 0.00071 % para `att-n48-k4`. Estos resultados sugieren que las instancias tomadas del TSP son instancias de dificultad media en comparación con los problemas propuestos directamente como instancias CVRP. Debido a las distancias tan grandes que aparecen en la instancia `att-n48-k4`, hemos utilizado valores $\lambda = \mu = 10000$ (10 veces mayor que en el resto de los casos) para evitar que posibles soluciones que no fueran correctas pudieran ser mejores que la óptima (véase la Sección 13.2).

13.5. Nuevas Soluciones al Problema VRP

Durante los estudios realizados en esta tesis, se han encontrado 10 nuevas mejores soluciones al problema VRP. Debido a la dificultad del problema y al elevado número de investigadores que trabajan en este problema en el mundo, este resultado supone un importante resultado para nuestro trabajo. Las instancias pertenecen a tres conjuntos de problemas diferentes, dos de los cuales se han estudiado en este capítulo (problemas de Van Breedam y Taillard), y el otro es el conjunto de problemas de gran escala (VLSVRP) estudiado en el Capítulo 10. En la Tabla 13.6 recopilamos las nuevas mejores soluciones que hemos encontrado. Nótese que en algunos casos la diferencia entre la nueva mejor solución encontrada y la mejor conocida previamente es muy pequeña, pero representan soluciones bien distintas.

Tabla 13.6: Nuevas mejores soluciones encontradas.

Inst.	**Nueva Mejor Solución**	**Δ (%)**	**Mejor Solución Anterior**
Bre-1	**1106.00**	3.24	1143.0 [281]
Bre-2	**1506.00**	4.68	1580.0 [281]
Bre-4	**1470.00**	0.41	1476.0 [281]
Bre-5	**950.00**	2.56	975.0 [281]
Bre-6	**969.00**	1.02	979.0 [281]
Bre-9	**1690.00**	5.59	1790.0 [281]
Bre-10	**1026.00**	1.82	1045.0 [281]
Bre-11	**1128.00**	2.76	1160.0 [281]
tai75b	**1344.62**	0.00	1344.64 [266]
vls26	**23977.73**	0.00	23977.74 [181]

13.6. Conclusiones

En este capítulo hemos desarrollado un único algoritmo que es capaz de competir con muchas técnicas de optimización distintas muy conocidas para resolver el CVRP. Nuestro algoritmo ha sido probado en un conjunto muy grande de 160 instancias con distintas características, por ejemplo, con clientes uniforme y no uniformemente distribuidos, con clientes agrupados y no agrupados, con un almacén en el centro o sin estar centrado, teniendo máxima distancia de rutas o no, considerando tiempos de entrega o no, con demandas homogéneas o heterogéneas, etc.

Consideramos que el comportamiento de nuestro algoritmo celular con la mutación combinada más la búsqueda local es muy satisfactorio ya que obtiene la mejor solución conocida en la mayoría de los casos, o valores muy cercanos a ella, para todos los problemas probados. Además, ha sido capaz de mejorar la mejor solución conocida hasta el momento para nueve de las instancias probadas, lo cual es una contribución importante a la investigación. Por tanto, podemos decir que el rendimiento de JCell2o1i es similar o incluso mejor que el del mejor algoritmo existente para cada instancia. Por otro lado, nuestro algoritmo es bastante simple ya que hemos diseñado un cGA canónico con tres mutaciones ampliamente utilizadas en la literatura para este problema, más dos métodos muy conocidos de búsqueda local.

Finalmente, queremos destacar los buenos resultados que hemos obtenido en esta tesis para el problema VRP ya que, además de las nueve nuevas mejores soluciones encontradas en este capítulo, debemos añadir la nueva mejor solución encontrada en el Capítulo 10 para el conjunto de problemas VLSVRP. Además, en el Capítulo 10 se ha encontrado la mejor solución conocida estimada (nunca antes encontrada por otro algoritmo) en seis de las doce instancias.

Capítulo 14

Caso Continuo: Conjunto Clásico de Problemas de Optimización Continua

En el Capítulo 5 hemos realizado una primera aproximación con el cGA para afrontar problemas en el espacio continuo de búsqueda. En ese capítulo, se aplica un cGA que utiliza una recombinación aritmética (AX) y una mutación uniforme (UM) a tres problemas de codificación real (las funciones Rastrigin, Ackley y Fractal), con resultados prometedores. La motivación de este capítulo es la de profundizar en el estudio del comportamiento de los cGAs cuando resuelven problemas de codificación real. Por eso, hemos seleccionado un gran conjunto de problemas usados normalmente en la literatura que tienen mayor tamaño (y por tanto mayor complejidad) que los utilizados en el Capítulo 5, y comparamos los resultados con algunas de las mejores aproximaciones que conocemos de la literatura. Una de las principales contribuciones de este capítulo es que mejoramos mucho los resultados mostrados en el Capítulo 5; además, nuestro algoritmo, un cGA canónico que implementa dos operadores de recombinación y mutación muy conocidos, obtiene resultados muy competitivos, o incluso mejores en muchos casos, que los (complejos) algoritmos con los que comparamos, pertenecientes al estado del arte en optimización continua.

La estructura del capítulo es la siguiente. Los resultados obtenidos, junto con la parametrización del algoritmo, se presentan en la Sección 14.1, y se comparan con otros algoritmos que pertenecen al estado del arte para estos mismos problemas. Finalmente, daremos nuestras principales conclusiones en la Sección 14.2.

14.1. Experimentación

En esta sección presentamos los resultados de resolver todos los problemas con JCell (nuestro cGA) y compararlos con los otros algoritmos del estado del arte. El conjunto de problemas utilizado en nuestro estudio es el mismo que el propuesto en [142], con el fin de poder comparar fácilmente nuestro algoritmo con otros pertenecientes al estado del arte en este campo. El conjunto de problemas está compuesto por seis funciones académicas clásicas y bien conocidas (Griewangk, Rastrigin, Rosenbrock, Schwefel,

Tabla 14.1: Parametrización.

Tamaño de la Población	144 individuos (12×12)
Vecindario	NEWS
Selección de los Padres	BT + BT
Recombinación	BLX-α; $\alpha = 0.5$
Probabilidad de Recombinación	$p_c = 1.0$
Mutación	Mutación No Uniforme
Probabilidad de Mutación	$p_m = 1/(2 \cdot n)$
Reemplazo	Reemp_si_no_Peor
Condición de Parada	800000 evaluaciones de fitness

Sphere, y ef_{10}), más tres problemas adicionales tomados del mundo real (FMS, SLE y Chebyschev). Todos estos problemas se encuentran descritos en el Apéndice A. Los problemas (de minimización) seleccionados están definidos por muy diversas características (lineal y no lineal, unimodal y multimodal, escalabilidad, convexidad, etc.), y nos permiten hacer posible nuestro propósito de comparación con otros algoritmos de la literatura. Por tanto, consideramos que esta batería de problemas es lo suficientemente amplia y diversificada para sustentar nuestras afirmaciones cuando evaluemos las aproximaciones algorítmicas (donde se presentan los algoritmos comparados).

En este trabajo utilizamos la misma parametrización para todos los problemas. Como se puede ver en la Tabla 14.1, usamos una población de 144 individuos, contenida en una rejilla bidimensional de 12×12 celdas. El vecindario utilizado es el llamado NEWS, de donde los dos padres son seleccionados utilizando *torneo binario*, forzándose que sean distintos.

Los descendientes se obtienen aplicando el *Operador de Recombinación por Aleación* BLX-α [98] (con $\alpha = 0.5$), a los dos padres seleccionados. El descendiente $\vec{z}$ generado por BLX-α está compuesto por los genes z_i escogidos aleatoriamente (de manera uniforme) del intervalo $[min(x_i, y_i) - I \cdot \alpha, max(x_i, y_i) + I \cdot \alpha]$, donde $I = max(x_i, y_i) - min(x_i, y_i)$, siendo x_i e y_i el gen *i-ésimo* de cada padre $\vec{x}$ y $\vec{y}$, respectivamente. La recombinación se lleva a cabo con probabilidad $p_c = 1.0$, y le aplicamos al descendiente resultante el operador de *Mutación No Uniforme*, considerado uno de los más adecuados para GAs de codificación real [143]. Este operador de mutación se aplica a los alelos del descendiente con una probabilidad $p_m = 1/(2 \cdot n)$, donde n es el tamaño del problema (longitud del cromosoma), y los muta utilizando una distribución no uniforme, cuyo tamaño de paso disminuye conforme avanza la ejecución. Por eso, hace una búsqueda uniforme en el espacio inicial (estado de exploración) y muy local para un estado más avanzado, favoreciendo así la explotación local. Finalmente, usamos una política de reemplazo elitista (*reemplazamos sólo si no es peor*), en la que el nuevo descendiente reemplaza al individuo actual para formar parte de la población de la siguiente generación sólo si no es peor (valor de fitness menor). El algoritmo para tras realizar 800000 evaluaciones, hemos utilizado el mismo límite que se usa en [142].

En los experimentos de este capítulo realizamos 50 ejecuciones independientes de cada problema y algoritmo. Con la intención de obtener confianza estadística en nuestra comparación, aplicamos el análisis de ANOVA o el de Kruskal-Wallis a los resultados (dependiendo de si los datos están uniformemente distribuidos o no, respectivamente). La distribución normal de los datos se comprueba utilizando el análisis de Kolmogorov-Smirnov. Al igual que en los capítulos anteriores, utilizamos Matlab(c) para este estudio de significancia estadística. Hemos incluido este tipo de análisis debido a la necesidad de formalismos matemáticos en los estudios actuales de metaheurística.

Tabla 14.2: Resultados computacionales. Mejores soluciones obtenidas.

Alg.	$\mathbf{f}_{\text{fms}}$	$\mathbf{f}_{\text{Gri}}$	$\mathbf{f}_{\text{Ras}}$	$\mathbf{f}_{\text{Ros}}$	$\mathbf{f}_{\text{Sch}}$	$\mathbf{f}_{\text{Sph}}$	$\mathbf{f}_{\text{ef}_{10}}$	$\mathbf{f}_{\text{SLE}}$	$\mathbf{f}_{\text{Cheb}}$
JCell	4.47E-27	**90 %**	**80 %**	1.331E0	**9.283E-161**	**1.718E-158**	**90 %**	**26 %**	1.284E3
$p_m = \frac{2}{n}$	2.946E-27	**78 %**	**72 %**	1.366E1	1.566E-36	8.136E-37	4.38E-17	**50 %**	1.249E3
$p_m = \frac{1}{n}$	9.139E-27	**80 %**	**68 %**	9.873E-2	1.63E-101	6.891E-98	2.198E-56	**30 %**	1.264E3
$\alpha = 0.25$	1.377E-4	3.035E-4	7.317E-5	1.746E1	2.306E-5	6.437E-8	1.575E-8	**66 %**	1.292E3
$\alpha = 0.75$	**3.511E-36**	**48 %**	**80 %**	**4.538E-3**	1.32E-100	4.446E-98	1.907E-71	**42 %**	**1.166E3**
p_c =0.50	4.456E-7	**2 %**	2.923E-7	9.917E-1	8.515E-30	6.342E-23	1.335E-6	**14 %**	1.225E3
p_c =0.75	1.25E-18	**60 %**	**40 %**	1.385E0	4.507E-134	1.298E-130	6.595E-80	**50 %**	1.284E3

Tabla 14.3: Resultados computacionales. Soluciones medias.

Alg.	$\mathbf{f}_{\text{fms}}$	$\mathbf{f}_{\text{Gri}}$	$\mathbf{f}_{\text{Ras}}$	$\mathbf{f}_{\text{Ros}}$	$\mathbf{f}_{\text{Sch}}$	$\mathbf{f}_{\text{Sph}}$	$\mathbf{f}_{\text{ef}_{10}}$	$\mathbf{f}_{\text{SLE}}$	$\mathbf{f}_{\text{Cheb}}$
JCell	9.06E0	**2.38E-3**	2.73E-5	1.78E1	**6.58E-158**	**1.50E-154**	**1.37E-3**	4.32E-10	1.39E3
	$\pm 6.65E0$	$\pm 4.75E-3$	$\pm 9.30E-6$	$\pm 1.09E1$	$\pm 2.18E-157$	$\pm 5.47E-154$	$\pm 5.80E-3$	$\pm 2.36E-9$	$\pm 4.80E1$
$p_m = \frac{2}{n}$	**2.11E0**	3.04E-3	9.11E-5	2.37E1	7.52E-35	2.17E-34	1.14E-2	1.57E-5	1.41E3
	$\pm 4.57E0$	$\pm 6.12E-3$	$\pm 2.82E-4$	$\pm 1.38E1$	$\pm 1.12E-34$	$\pm 5.13E-34$	$\pm 2.97E-2$	$\pm 1.11E-4$	$\pm 6.14E1$
$p_m = \frac{1}{n}$	4.07E0	3.53E-3	**7.31E-6**	2.14E1	2.30E-98	1.22E-95	4.83E-3	4.84E-10	1.40E3
	$\pm 5.79E0$	$\pm 5.40E-3$	$\pm 2.97E-5$	$\pm 1.36E1$	$\pm 9.13E-98$	$\pm 4.25E-95$	$\pm 1.15E-2$	$\pm 3.42E-9$	$\pm 5.84E1$
$\alpha = 0.25$	7.66E0	1.54E-2	4.92E-4	2.27E1	9.67E-2	2.09E-6	9.68E-2	2.38E-5	1.42E3
	$\pm 6.49E0$	$\pm 1.37E-2$	$\pm 5.04E-4$	$\pm 7.45E0$	$\pm 9.67E-2$	$\pm 1.72E-6$	$\pm 1.05E-1$	$\pm 1.69E-4$	$\pm 5.71E1$
$\alpha = 0.75$	7.75E0	8.78E-3	3.17E-5	**1.13E1**	2.87E-97	4.87E-95	3.85E-3	**1.78E-14**	**1.38E3**
	$\pm 7E0$	$\pm 1.04E-2$	$\pm 9.90E-5$	$\pm 1.35E1$	$\pm 8.31E-97$	$\pm 2.68E-94$	$\pm 9.98E-3$	$\pm 1.68E-14$	$\pm 6.65E1$
$p_c = 0.50$	7.98E0	8.96E-3	1.01E-4	3.30E1	1.84E-3	2.73E-8	2.96E-1	4.47E-5	1.39E3
	$\pm 7.01E0$	$\pm 1.28E-2$	$\pm 1.34E-4$	$\pm 2.67E1$	$\pm 1.07E-2$	$\pm 8.18E-8$	$\pm 4.96E-1$	$\pm 2.96E-4$	$\pm 6.15E1$
p_c =0.75	9.41E0	6.81E-3	2.85E-5	2.67E1	2.16E-10	2.65E-14	3.38E-2	1.49E-7	1.39E3
	$\pm 7.46E0$	$\pm 9.7E-3$	$\pm 8.53E-5$	$\pm 2.07E1$	$\pm 1.53E-9$	$\pm 1.87E-13$	$\pm 5.81E-2$	$\pm 8.85E-7$	$\pm 4.69E1$
Test	+	+	+	+	+	+	+	+	+

14.1.1. Ajustando el Algoritmo

En esta sección procederemos al ajuste de los valores de algunos parámetros del cGA para mejorar el comportamiento del algoritmo en la medida de lo posible. La base de la parametrización está descrita en la Tabla 14.1 (algoritmo JCell), y probamos el algoritmo con dos probabilidades de mutación distintas ($p_m = 2/n$ y $p_m = 1/n$), dos valores diferentes para α ($\alpha = 0.25$ y $\alpha = 0.75$), y dos probabilidades de recombinación ($p_c = 0.5$ y $p_c = 0.75$). Los mejores valores obtenidos para cada problema con los 7 cGAs se muestran en la Tabla 14.2. Si el óptimo global se ha localizado durante algunas ejecuciones, presentaremos el porcentaje de ejecuciones en los que esto ocurre (en este caso, el símbolo '%' aparece en la tabla justo detrás del valor). Además, la media de los resultados obtenidos, junto con la desviación típica se muestran en la Tabla 14.3. Los mejores valores de las dos tablas para cada problema están en **negrita**.

En la Tabla 14.2, podemos observar que JCell (utilizando la parametrización de la Tabla 14.1) es el más preciso de todos los algoritmos probados ya que obtiene los mejores resultados para 6 de los 9 problemas de test. Además, JCell también obtiene el mejor comportamiento de todos en términos de valor medio de fitness (en las 50 ejecuciones) en 4 de los 9 problemas, como podemos ver en la Tabla 14.3. En los otros 5 problemas, los resultados presentados por JCell, tienen el mismo orden de magnitud que los mejores valores de los cGAs que estamos comparando, con excepción del f_{SLE}. Existe confianza estadística en los resultados de los cGAs comparados en todos los problemas.

A partir de las Tablas 14.2 y 14.3 podemos notar que el cambio en las probabilidades de recombinación y mutación, así como en los valores de α de la parametrización del algoritmo propuesto en la Tabla 14.1 no nos conduce a ninguna mejora en el comportamiento del algoritmo, en general. Las mejoras más notables son las obtenidas para f_{Ros} y f_{SLE} cuando utilizan $\alpha = 0.75$, y f_{fms} al aumentar la probabilidad de mutación a $2/n$. De estos tres casos, sólo f_{fms} con $p_m = 2/n$ es mejor que el JCell con confianza estadística. Por tanto, podemos concluir que JCell es el mejor de todos los algoritmos propuestos y por eso lo seleccionamos, en la siguiente sección, para nuestras comparaciones con otras aproximaciones de la literatura.

Tabla 14.4: Resultados computacionales. Media del tiempo de ejecución (en segundos).

Alg.	$\mathbf{f}_{\text{fms}}$	$\mathbf{f}_{\text{Gri}}$	$\mathbf{f}_{\text{Ras}}$	$\mathbf{f}_{\text{Ros}}$	$\mathbf{f}_{\text{Sch}}$	$\mathbf{f}_{\text{Sph}}$	$\mathbf{f}_{\text{ef}_{10}}$	$\mathbf{f}_{\text{SLE}}$	$\mathbf{f}_{\text{Cheb}}$
JCell	**132.79**	**7.39**	**9.63**	24.87	**22.43**	18.31	24.93	10.41	540.96
	±2.90	±7.70	±5.46	±0.22	±0.20	±0.18	±0.37	±5.63	±1.05
$p_m = \frac{2}{n}$	140.17	15.97	19.08	**24.69**	23.49	19.26	27.16	7.35	**484.21**
	±1.69	±4.45	±2.65	±0.34	±0.16	±0.16	±0.13	±4.14	±0.82
$p_m = \frac{1}{n}$	137.74	9.87	11.94	29.61	26.97	17.70	27.13	7.70	565.67
	±2.71	±0.18	±0.11	±0.21	±0.25	±0.14	±0.25	±4.40	±1.78
$\alpha = 0.75$	135.68	11.84	10.60	29.73	25.84	17.47	26.40	6.88	564.40
	±3.58	±8.13	±6.06	±0.54	±0.14	±0.19	±0.07	±3.96	±1.89
$p_c = 0.50$	135.57	17.86	17.52	27.03	23.47	**14.97**	**23.72**	7.40	562.64
	±3.35	±2.75	±0.10	±0.41	±0.11	±0.11	±0.21	±2.41	±1.33
$p_c = 0.75$	139.05	11.11	14.63	28.24	25.14	16.21	24.22	7.07	563.36
	±2.97	±8.16	±5.40	±0.47	±0.14	±0.22	±0.64	±4.50	±2.34
Test	+	+	+	+	+	+	+	+	+

Para poder realizar un estudio más profundo de los algoritmos propuestos, también incluimos una comparación de la media de los tiempos de ejecución (en segundos) de los algoritmos en la Tabla 14.4 para los problemas que se proponen (el mejor –más bajo– resultado de cada problema está en **negrita**). Como podemos observar, existen diferencias significativamente estadísticas para todos los problemas en los resultados que se presentan. JCell es el más rápido de los algoritmos comparados en 4 de los 9 problemas (se encuentran diferencias significativas para f_{fms}, f_{Gri}, y f_{Sch}), mientras que sólo existen algoritmos más rápidos que JCell con diferencia significativa en 3 de todos los problemas estudiados (f_{Sph}, $f_{\text{ef}_{10}}$, y f_{SLE}). Las diferencias más importantes de los valores mostrados en la Tabla 14.4 son debidos a las distintas tasas de éxito obtenidas, ya que el algoritmo se detiene antes de alcanzar las 800000 evaluaciones al encontrar el óptimo. Por ejemplo, JCell es entre 2 y 3 veces más rápido que el algoritmo con $\alpha = 0.25$ cuando ambos algoritmos encuentran el óptimo, lo que se contradice con los resultados de la Tabla 14.4, que muestra en media 10.41 segundos para JCell y 5.65 segundos para el otro algoritmo (con $\alpha = 0.25$). La razón es que éste último encuentra el óptimo en el 66 % de las ejecuciones mientras que la tasa de éxito del primero es sólo del 26 % (véase la Tabla 14.2).

14.1.2. Comparación con Otros Algoritmos

En esta sección comparamos JCell con otros algoritmos del estado del arte en optimización continua. Estos algoritmos son los GAs graduales distribuidos de codificación real, propuestos el en trabajo de Herrera y Lozano [142]. Los problemas estudiados en este trabajo son los mismos de nuestro conjunto de problemas, y también la dimensión del espacio de búsqueda utilizado en cada problema (10 variables para $f_{\text{ef}_{10}}$ y 25 para el resto de los problemas). Además, la condición de parada de los algoritmos comparados es la misma que utilizamos en este trabajo: alcanzar 800000 evaluaciones de la función de fitness.

Los tres algoritmos comparados [142] son GAs heterogéneos distribuidos de codificación real que se diferencian en el operador de recombinación utilizado. Se basan en una topología hipercúbica de tres dimensiones, y en cada vértice del cubo hay una población panmíctica de 20 individuos. Cada sub-población tiene una parametrización del operador de recombinación distinta, de forma que las poblaciones que están en la cara delantera del cubo se especializan en la exploración, mientras que el operador de recombinación incentiva la explotación en la cara posterior. Con una frecuencia dada, se producen migraciones de individuos entre poblaciones que pertenecen a una misma cara del cubo. Los tres GAs distribuidos heterogéneos que se presentan en [142] son GD-FCB, el cual utiliza el *Operador de Recombinación Basado en Conectivas de Lógica Difusa* (FCB) [141], GD-BLX, implementando el *Operador de Recombinación por Aleación* (BLX-α) [98], y GD-EFR, que utiliza el *Operador de Recombinación de Lógica Difusa Extendido* (EFR) [283]. Una descripción más completa del modelo de los GAs heterogéneos distribuidos de codificación real puede encontrarse en [142].

Tabla 14.5: Comparación con otras aproximaciones: Problemas académicos. Mejor solución encontrada.

Algoritmo	$\mathbf{f}_{\text{Gri}}$	$\mathbf{f}_{\text{Ras}}$	$\mathbf{f}_{\text{Ros}}$	$\mathbf{f}_{\text{Sch}}$	$\mathbf{f}_{\text{Sph}}$	$\mathbf{f}_{\text{ef}_{\mathbf{10}}}$	$\mathbf{f}_{\text{fms}}$	$\mathbf{f}_{\text{SLE}}$	$\mathbf{f}_{\text{Cheb}}$
JCell	**90 %**	80 %	1.3E0	**9.3E-161**	**1.7E-158**	**90 %**	4.5E-27	**26 %**	1.3E3
GD-FCB	4E-11	9E-12	**3E-5**	6E-1	4E-14	8E-4	7E-26	3E0	4E1
GD-BLX	60.0 %	**100 %**	2E1	7E-8	8E-58	5E-48	**66.7 %**	2E0	**1E1**
GD-EFR	53.3 %	**100 %**	1E1	2E-7	8E-51	6E-47	43.3 %	9E-1	**1E1**

Tabla 14.6: Comparación con otras aproximaciones: Problemas académicos. Media de las soluciones encontradas.

Algoritmo	$\mathbf{f}_{\text{Gri}}$	$\mathbf{f}_{\text{Ras}}$	$\mathbf{f}_{\text{Ros}}$	$\mathbf{f}_{\text{Sch}}$	$\mathbf{f}_{\text{Sph}}$	$\mathbf{f}_{\text{ef}_{\mathbf{10}}}$	$\mathbf{f}_{\text{fms}}$	$\mathbf{f}_{\text{SLE}}$	$\mathbf{f}_{\text{Cheb}}$
JCell	**2.4E-3**	2.7E-5	1.8E1	**6.6E-158**	**1.5E-154**	1.4E-3	9.1E0	**4.3E-10**	1.4E3
GD-FCB	2E-2	4E-11	**9E0**	4E0	2E-13	2E-3	1E1	4E1	**2E2**
GD-BLX	5E-3	**0E0**	2E1	2E-6	8E-53	**6E-36**	**3E0**	3E1	**2E2**
GD-EFR	7E-3	**0E0**	2E1	4E-6	4E-47	9E-35	6E0	2E1	**2E2**

Los resultados de todos estos algoritmos, junto con los resultados de JCell, se muestran en las tablas 14.5 (mejores valores encontrados) y 14.6 para la media de los valores encontrados por los algoritmos.

Si prestamos atención a las mejores soluciones encontradas por el algoritmo (Tabla 14.5), podemos ver que JCell obtiene los mejores resultados para 5 de los 9 problemas. Además, en el caso de $f_{\text{ef}_{10}}$, JCell encuentra la solución óptima en el 90 % de las ejecuciones, mientras que las otras tres aproximaciones no son capaces de encontrar el óptimo en ninguna ejecución (la mejor aproximación es 5E-48 para GD-BLX). También para f_{SLE} JCell es el único algoritmo que encuentra la óptima solución al problema, y lo encuentra en el 26 % de las ejecuciones. La diferencia entre JCell y el mejor de los algoritmos comparados en el caso de f_{Ras} es pequeña, ya que encuentra el óptimo en el 80 % de las soluciones frente al 100 % de GD-BLX y GD-EFR.

En referencia a la media de las soluciones encontradas, JCell mejora las otras aproximaciones en 4 de las 9 funciones (Tabla 14.6), con una importante diferencia de 150 órdenes de magnitud para f_{Sch} y f_{Sph} (en media y mejores soluciones) y 11 órdenes de magnitud para f_{SLE} con respecto a los mejores valores presentados por los otros algoritmos. Además, no consideramos importantes las diferencias existentes entre JCell, GD-BLX, y GD-EFR para f_{Ras}, ya que JCell encuentra el óptimo global en el 80 % de las ejecuciones (mientras que GD-BLX y GD-EFR lo hacen en el 100 %). En el caso de $f_{\text{ef}_{10}}$, el valor medio tan elevado que presenta JCell en este problema es debido a que se estanca en óptimos locales (el valor ronda 10^{-2}) en el 10 % de las ejecuciones. Finalmente, la solución media encontrada por JCell para f_{Ros} está casi en el mismo orden de magnitud del mejor algoritmo, GD-FCB.

En cuanto a las funciones tomadas del mundo real, en el caso de f_{fms}, el resultado medio obtenido por todos los algoritmos es del mismo orden de magnitud, y la mejor solución encontrada por JCell es cercana al óptimo (4.5E-27), aunque GD-BLX y GD-EFR encuentran la solución óptima en el 66.7 % y 43.3 % de las ejecuciones, respectivamente. Finalmente, f_{Cheb} es el único problema para el que JCell es el peor de los algoritmos comparados, tanto en términos de la mejor solución encontrada como de la solución media. De todas maneras, las diferencias entre JCell y los otros algoritmos es sólo de un orden de magnitud en la media de las soluciones encontradas.

Finalmente, podemos resumir estos resultados concluyendo que JCell mejora claramente a las tres aproximaciones heterogéneas distribuidas en 5 de las 9 instancias probadas, mejorando el estado del arte actual, y está muy cerca de los resultados de los mejores algoritmos en los otros problemas. Tal vez, la excepción sea f_{Cheb}, para el que JCell tiene los peores resultados de los algoritmos comparados, aunque

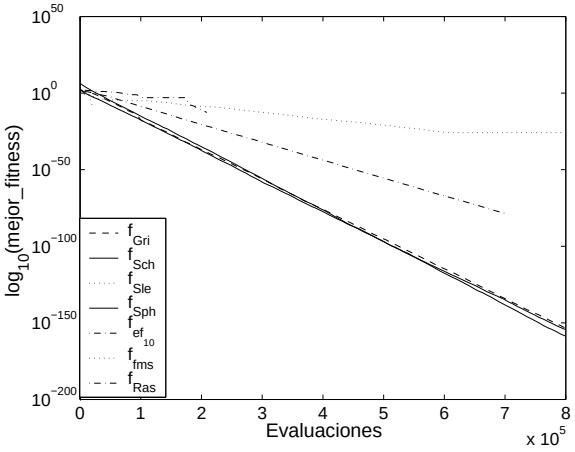


Figura 14.1: Evolución del valor medio de fitness para f_{Gri}, f_{Sch}, f_{SLE}, f_{Sph}, $f_{\text{ef}_{10}}$, f_{fms} y f_{Ras}.

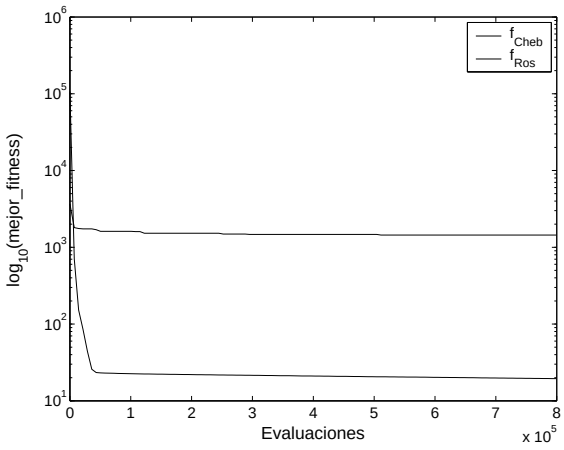


Figura 14.2: Evolución del valor medio de fitness para f_{Cheb}, y f_{Ros}.

las diferencias sean de sólo un orden de magnitud.

En este punto, nos gustaría enfatizar la simplicidad de nuestra aproximación frente a los otros tres algoritmos distribuidos. Estamos utilizando en este trabajo un cGA canónico para resolver los problemas, mientras que los algoritmos comparados son cGAs heterogéneos distribuidos compuestos por 8 sub-poblaciones diferentes, cada una de las cuales tienen distintas parametrizaciones, con el consecuente incremento en los parámetros que necesita el algoritmo. Además, nuestra aproximación usa exactamente los mismos operadores de recombinación y mutación que implementa el GD-BLX, por lo que la diferencia en el rendimiento de los dos algoritmos es intrínseco al modelo del algoritmo. Por tanto, la relación de exploración/explotación que realiza el cGA canónico en la población, conseguida simplemente introduciendo el concepto de vecindario en una rejilla estructurada de individuos, consigue resultados similares e incluso mejores en muchos casos, con respecto al complejo modelo de hipercubo que propusieron Herrera y Lozano en [142].

En las Figuras 14.1 y 14.2 dibujamos la evolución del valor medio de fitness de los individuos en la población durante ejecuciones típicas de JCell para todos los problemas estudiados. Observe que este valor medio de fitness (eje de ordenadas) está en escala logarítmica. Para la funciones representadas en la Figura 14.1, se puede ver en general una aproximación rápida hacia la mejor solución del problema durante la ejecución, lo que significa una convergencia progresiva de la población. Entre estos problemas, se comprueba la dificultad de f_{fms}, que muestra la convergencia más lenta. Tras incrementar lo suficiente el número máximo de evaluaciones permitidas, hemos comprobado que esta convergencia se mantiene hasta que se encuentra el óptimo en la mayoría de las ejecuciones para las funciones de la Figura 14.1. La excepción es f_{fms}, que en algunas ejecuciones cae en óptimos locales de los que no puede escapar.

Los problemas más difíciles para JCell de entre los que hemos estudiado en este trabajo son f_{Cheb}, y f_{Ros}. Mostramos estos dos problemas en la Figura 14.2. Se puede observar cómo el algoritmo experimenta una convergencia realmente rápida al principio de la búsqueda (podemos comprobar que el valor medio de fitness se dibuja en escala logarítmica) pero, después de eso, se estanca en óptimos locales, la diversidad de la población se pierde rápidamente, y por tanto no vuelve a haber mejoras importantes en las soluciones.

14.2. Conclusiones

Hemos propuesto en este capítulo una nueva aproximación (utilizando JCell) para resolver problemas en un espacio de búsqueda continuo. Nuestra propuesta consiste en un sencillo cGA canónico que utiliza dos operadores de recombinación y mutación muy conocidos en el área de los GAs de codificación real.

Los resultados mostrados en esta primera aproximación son realmente prometedores, ya que JCell mejora algoritmos heterogéneos distribuidos ya existentes, y realmente complejos, que pertenecen al estado del arte. Además, uno de esos algoritmos, GD-BLX, implementa exactamente los mismos operadores de recombinación y mutación que utilizamos en JCell, lo que destaca la buena relación de exploración/explotación que el cGA realiza sobre la población simplemente utilizando el concepto de vecindario. En el caso de los tres GAs heterogéneos que comparamos, esta relación entre la exploración y la explotación se realiza distribuyendo la población en pequeñas sub-poblaciones con distintas parametrizaciones en un hipercubo, y restringiendo el intercambio de individuos (migración) entre las sub-poblaciones que se localizan en la misma cara del hipercubo. Viendo los resultados obtenidos, podemos concluir que el complejo modelo de hipercubo de los tres algoritmos comparados no muestra una relación de exploración/exploitación mejor que JCell, el cual se basa en conceptos mucho más sencillos que el modelo de hipercubo y necesita establecer un número menor de parámetros (cada sub-población del hipercubo implementa una parametrización distinta).

Capítulo 15

Caso Real: Optimización del Proceso de Difusión en MANETs

15.1. Introducción

Las redes móviles ad hoc (MANETs) son redes fluctuantes compuestas por un conjunto de dispositivos de comunicación llamados *nodos* (o *dispositivos*) que pueden conectarse espontáneamente entre ellos sin la necesidad de que exista ninguna infraestructura previa. Esto significa que no hay presente ninguna organización en estas redes como suele ser habitual en las redes de comunicación. Las tecnologías de redes inalámbricas más conocidas disponibles hoy en día para construir MANETs son Bluetooth e IEEE 802.11 (WiFi). Esto implica que a) los dispositivos se comunican con un alcance limitado, y b) los dispositivos se pueden mover mientras están en comunicación. Una consecuencia de esta movilidad es que la topología de estas redes puede cambiar rápidamente y de manera impredecible. Precisamente, este comportamiento dinámico constituye uno de los principales obstáculos para conseguir comunicaciones eficientes.

En este capítulo estudiamos el problema de difusión (o *broadcasting*) de una subclase particular de MANETs llamadas MANETs *Metropolitanas*, que tienen unas propiedades específicas: la densidad en las redes es heterogénea y dinámica (particularmente, las regiones con alta densidad no se mantienen activas todo el tiempo). La estrategia de difusión que consideramos en este trabajo es el protocolo llamado *Inundación con Retrasos y Vecindarios Acumulativos* (o *Delayed Flooding with Cumulative Neighborhood* –DFCN) [148]. Este problema se define por primera vez en este trabajo, y se han propuesto tres ejemplos (instancias del problema) de casos reales de este tipo de redes, como son un centro comercial, un área metropolitana y un entorno de autopistas, por lo que, en lugar de proporcionar un protocolo de propósito múltiple, la originalidad de nuestra propuesta reside en ajustar el servicio de difusión para cada red en particular. Optimizar una estrategia de difusión implica que hay que satisfacer múltiples objetivos a la vez, como maximizar el número de dispositivos alcanzados (cobertura), minimizar la utilización de red (ancho de banda) y/o minimizar la duración del proceso. Por eso, se conoce como problema de optimización multi-objetivo [60, 78].

La característica principal de la optimización multi-objetivo es que no se limita a encontrar una única solución como en la optimización de un sólo objetivo sino un conjunto de soluciones conocido como el *conjunto óptimo de Pareto*. La razón es que, tomando como ejemplo el problema con el que estamos tratando, una solución puede representar el mejor resultado considerando el número de dispositivos alcanzados (cobertura), mientras que otra solución distinta podría ser la mejor en términos de la duración

del proceso de difusión. Estas soluciones se dice que son *no dominadas*. El resultado que aporta un algoritmo de optimización multi-objetivo es un conjunto de soluciones no dominadas (el *conjunto óptimo de Pareto*) las cuales, cuando se representan en el espacio de objetivos, forman el llamado frente de *Pareto*. La misión del diseñador, experto en el dominio del problema, es escoger la solución más adecuada del frente de Pareto. Aquí proponemos un algoritmo evolutivo celular multi-objetivo (cMOGA), desarrollado específicamente para este estudio, para resolver el problema multi-objetivo de ajustar una estrategia particular de difusión para MANETs Metropolitanas.

Muchos de los algoritmos más populares para resolver problemas de optimización multi-objetivo son algoritmos evolutivos (EAs) [60, 78]. Sin embargo, muy pocos trabajos usan algoritmos genéticos basados en modelos celulares [163, 178, 212], aún cuando los algoritmos genéticos celulares (cGAs) han probado, en el pasado, tener una eficiencia y precisión muy alta en optimización mono-objetivo [15, 20, 22, 116]. El algoritmo que proponemos, cMOGA, es una contribución en este campo. Es más, según nuestro conocimiento, este trabajo es el primer intento realizado para resolver el problema de difusión de MANETs utilizando un EA multi-objetivo.

El resto del capítulo se organiza de la siguiente manera. En la Sección 15.2 se describe el problema que tratamos, el conjunto de escenarios propuestos y la estrategia de difusión que nos proponemos optimizar. Detallaremos la aproximación propuesta basada en un algoritmo genético celular multi-objetivo en la Sección 15.3. La Sección 15.4 presenta nuestro conjunto de experimentos y analiza los resultados. Compararemos el comportamiento de cMOGA con otros EAs que hemos aplicado al problema considerado en la Sección 15.5. El capítulo termina con nuestras principales conclusiones y unas líneas futuras de investigación.

15.2. El Problema

El problema que estudiamos en este capítulo consiste en, dada una red MANET metropolitana de entrada, determinar los parámetros más adecuados para la estrategia de difusión. Primero, describimos en la Sección 15.2.1 las características de la red objetivo que usamos. La Sección 15.2.2 está dedicada a la presentación de DFCN [148], la estrategia de difusión que tenemos que ajustar. Finalmente, los MOPs que definimos y estudiamos en este trabajo se presentan en la Sección 15.2.3.

15.2.1. Redes Móviles Ad Hoc Metropolitanas. El Simulador Madhoc

Las redes móviles ad hoc metropolitanas son MANETs con algunas propiedades particulares. En primer lugar, tienen una o más áreas donde la densidad de los nodos es mayor que la media. Se denominan *áreas de alta densidad* y se pueden detectar estadísticamente. Un área de alta densidad puede ser, por ejemplo, un supermercado, un aeropuerto o una oficina. En segundo lugar, las áreas de alta densidad no se mantienen activas todo el tiempo, es decir, pueden aparecer y desaparecer de la red (por ejemplo, un supermercado está abierto de 9 de la mañana a 10 de la noche, y fuera de ese periodo, la densidad de nodos en el área correspondiente es casi cero). Un ejemplo de red metropolitana de 4 km^2 con 2000 dispositivos se muestra en la Figura 15.1.

Para tratar con este tipo de redes, no hay otra solución que recurrir a simulaciones software. En este trabajo hemos utilizado *Madhoc*[1] [147], un simulador de una MANET metropolitana. Su objetivo es proporcionar una herramienta para simular diferentes niveles de servicios basados en distintas tecnologías de MANETs para distintos entornos.

[1] El simulador Madhoc está disponible de manera gratuita en `http://www-lih.univ-lehavre.fr/~hogie/madhoc/`

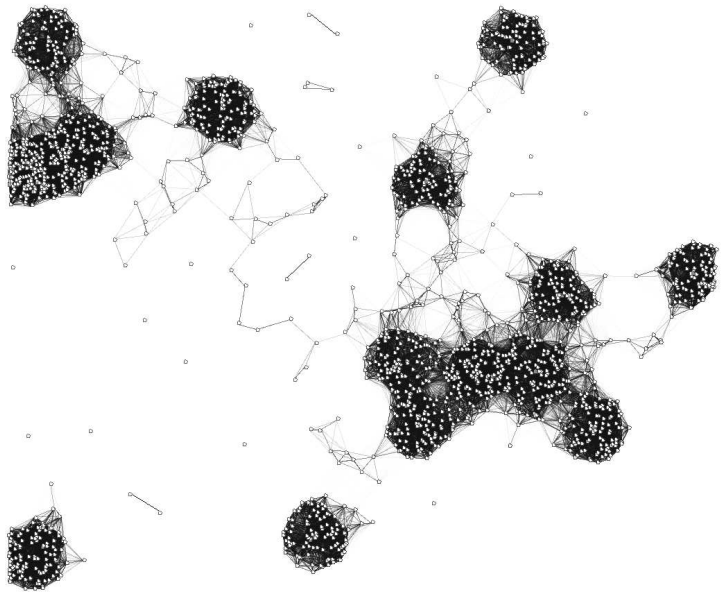


Figura 15.1: Un ejemplo de MANET metropolitana.

En el contexto de las MANETs metropolitanas se suelen encontrar varias configuraciones topológicas. Algunos ejemplos son redes construidas espontáneamente por gente moviéndose en lugares concretos, tales como mercados, estaciones de trenes, centros comerciales y barrios de ciudades. Todos estos escenarios tienen varias características distintas, como la movilidad y la densidad de los dispositivos, el tamaño del área y la presencia o no de paredes (que son obstáculos para movilidad y atenúan la fuerza de la señal), entre otras. Por tanto, en este trabajo se utilizan tres escenarios realistas muy distintos, implementados todos ellos en Madhoc. Estos escenarios se corresponden con los siguientes entornos del mundo real: un centro comercial, un área metropolitana y el entorno de una autopista.

- **Entorno de Centro Comercial.** El entorno de un centro comercial se propone para emular MANETs en superficies comerciales, en las que se suelen situar tiendas juntas unas con otras en pasillos. La gente va de una tienda a otra a través de esos pasillos, parando ocasionalmente para mirar algunos escaparates. Estos centros comerciales suelen tener una elevada concentración de personas (alta densidad de dispositivos), y la gente se comporta de manera distinta (en términos de movilidad) cuando están dentro o fuera de las tiendas. Además, podemos encontrar una alta densidad de tiendas en este tipo de escenarios. Finalmente, en el entorno del centro comercial tanto la movilidad de los dispositivos como la señal de propagación están restringidos por las paredes del edificio.

- **Entorno Metropolitano.** El segundo escenario realista que proponemos es el entorno metropolitano. Para ello, situamos un conjunto de puntos (cruces de calles) en la superficie modelada, que son unidos mediante calles. En este caso, modelamos tanto peatones como vehículos moviéndose continuamente de un cruce a otro. Como en el mundo real, los dispositivos deben reducir su velocidad cuando se acercan a un cruce. En este escenario, la obstrucción de las paredes en la fuerza de la señal será mayor que en el caso del entorno del centro comercial.

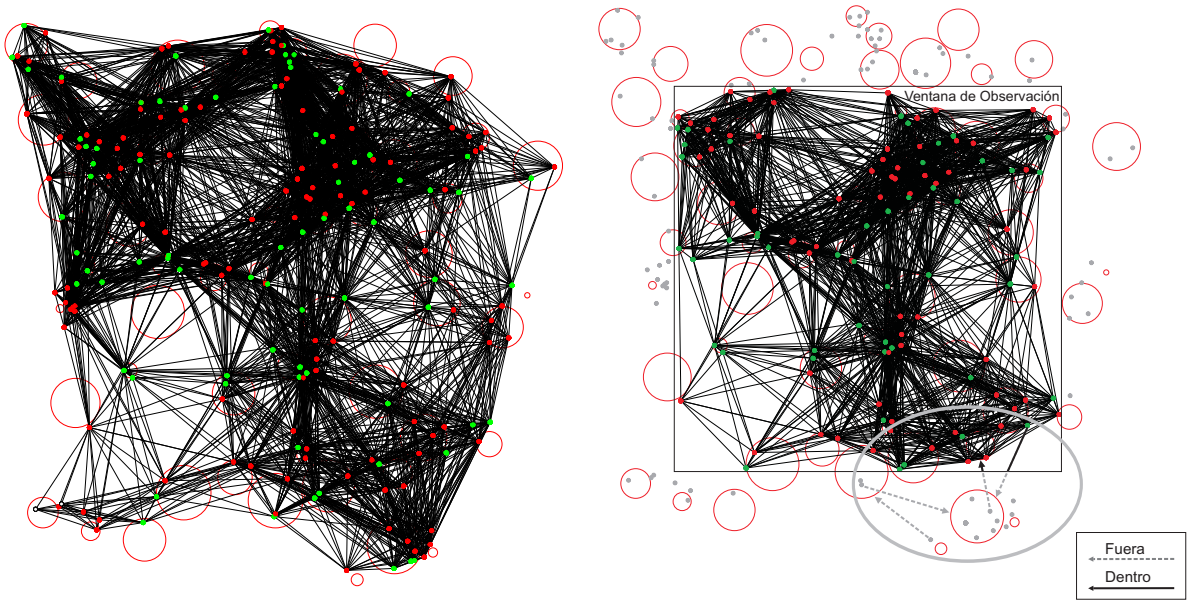


Figura 15.2: Los efectos de introducir una ventana de observación en la MANET estudiada.

- **Entorno de Autopista.** Utilizamos este entorno para simular el comportamiento de las MANETs fuera de las ciudades. Por ejemplo, pensemos en una gran superficie con carreteras y gente viajando en coche. En este contexto, debería haber una baja densidad de dispositivos por kilómetro cuadrado (todos los dispositivos se sitúan en las carreteras), moviéndose todos ellos muy rápido. Además, no existen obstáculos que atenúen la fuerza de la señal en las comunicaciones.

Para conseguir hacer nuestros estudios más realistas, incluimos en las simulaciones una *ventana de observación*, que supone una proporción del espacio de simulación, de manera que sólo se tienen en cuenta para las medidas los dispositivos situados dentro de esta ventana. Esto hace posible la simulación de los nodos entrando y saliendo de la red cuando llegan o abandonan la ventana de observación, respectivamente. Por tanto, permitimos la existencia de un número variable de dispositivos en la red, como sucede en las MANETs reales. En todas nuestras pruebas en este trabajo, esta ventana de observación cubre el 70 % del área total. Por ejemplo, en la Figura 15.2 podemos observar una MANET simulando un entorno de un centro comercial (a la izquierda) y la ventana de observación que estudiamos (a la derecha); suponiendo el 70 % de la totalidad de la red. Los círculos representan tiendas, mientras que los puntos son dispositivos (los que están fuera de la ventana de observación aparecen en gris, lo que quiere decir que no se consideran al realizar las medidas).

15.2.2. Inundación con Retrasos y Vecindarios Acumulativos

Williams y Camp [292] y, más recientemente, Stojmenovic y Wu [263] propusieron dos de los análisis de protocolos de difusión más referenciados. En su propuesta, Stojmenovic y Wu [263] constataron que los protocolos se pueden clasificar según la naturaleza de su algoritmo –determinismo (no hace uso de la aleatoriedad), fiabilidad (garantía de cobertura total)– o según la información requerida para sus ejecuciones (información de red, mensajes de contenido “hola”, mensajes de contenido broadcast). De

manera similar, Wu y Lou [296] clasifican los protocolos como *centralizados* [228] y *localizados*. Los protocolos centralizados necesitan tener un conocimiento global o casi global de la red, por lo que no son escalables. Los protocolos localizados son aquellos que necesitan conocimiento del vecindario a 1 ó 2 saltos, siendo el vecindario a 1 salto de un dispositivo dado el conjunto de dispositivos visibles directamente por él, y el vecindario a 2 saltos es este vecindario mas los vecindarios de sus vecinos.

Utilizando estas clasificaciones, la inundación con retrasos y vecindarios acumulativos (DFCN) [148] es un algoritmo determinista. No consiste en una nueva aproximación para calcular conjuntos dominantes, sino en un protocolo totalmente localizado que define heurísticas basadas en información en 1-salto. Esto permite a DFCN tener gran escalabilidad. Los mensajes "hola"que intercambian los nodos para conocer su vecindario en todo momento no llevan información adicional. Sólo los mensajes de difusión deben incrustarse en la lista de los nodos vecinos.

Para poder ejecutar el protocolo DFCN, debemos asumir lo siguiente:

- Como muchos otros protocolos de difusión que se basan en el conocimiento del vecino (FWSP, SBA, etc.) [184, 231], DFCN necesita conocer el vecindario de 1-salto. Esto se consigue utilizando los paquetes "hola" en una capa de red más baja. El conjunto de vecinos del dispositivo s se llama $N(s)$.

- Cada mensaje m contiene –incluido en su cabecera– el conjunto de identificadores (IDs) de los vecinos a 1-salto de sus remitentes más recientes.

- Cada dispositivo mantiene información local sobre todos los mensajes recibidos. Cada instancia de esta información local está formada por los siguientes puntos:

  - el ID del mensaje recibido;
  - el conjunto de IDs de los dispositivos que se sabe que han recibido el mensaje;
  - la decisión de si el mensaje debe reenviarse o no.

- DFCN necesita utilizar un retraso aleatorio antes de re-emitir un mensaje de difusión m. Este retraso, llamado *Retraso de Estimación Aleatoria (RAD)*, está previsto para prevenir colisiones. Para ser más exactos, cuando un dispositivo s envía un mensaje m, todos los dispositivos de $N(s)$ lo reciben a la vez. Es probable entonces que todos ellos reenvíen m simultáneamente, y esta simultaneidad implica colisiones de red. El objetivo de RAD es retrasar aleatoriamente la retransmisión de m en cada nodo. Como cada dispositivo de $N(s)$ espera a que termine un RAD distinto antes del reenvío de m, el riesgo de colisiones se reduce considerablemente.

DFCN es un algoritmo orientado a eventos que puede dividirse en tres partes principales: las dos primeras tienen el objetivo de manejar los eventos que llegan, que son (i) recepción de un nuevo mensaje y (ii) detección de un nuevo vecino. La tercera parte (iii) consiste en tomar la decisión de emitir como respuesta a uno de los dos eventos anteriores. El comportamiento resultante de la recepción de un mensaje se denomina comportamiento *reactivo*; en cambio, cuando se descubre un nuevo vecino, el comportamiento seguido se denomina comportamiento *proactivo*.

Consideremos que s_1 y s_2 son dos dispositivos cada uno en el vecindario a 1 salto del otro. Cuando s_1 envía un paquete a s_2, le añade al paquete la información sobre el conjunto $N(s_1)$. Al recibir, s_2 por tanto sabe que cada dispositivo en $N(s_1)$ ha recibido el paquete. El conjunto de vecinos de s_2 que *potencialmente* no ha recibido todavía el paquete es entonces $N(s_2) - N(s_1)$. Si s_2 reemite el

paquete, el número efectivo de nuevos dispositivos alcanzados se maximiza por la función: $h(s_2,\ s_1)\ =\ |N(s_2)\ -\ N(s_1)|$.

Para minimizar el uso de la red causado por el posible reenvío de un paquete, un mensaje será reenviado únicamente si el número de nuevos dispositivos es mayor que un umbral dado. Este umbral se establece en función del número de dispositivos en el vecindario (la densidad de red local) del dispositivo receptor s_2, y se escribe "threshold($|N(s)|$)". La decisión que toma s_2 de reenviar el paquete recibido de s_1 se define por una función booleana:

$$B(s_2, s_1) = \begin{cases} \text{verdadero} & \text{si } h(s_2, s_1) \geq \text{threshold}(|N(s_2)|) \\ \text{falso} & \text{en otro caso} \end{cases}.$$

Si se excede el umbral, el dispositivo receptor s_2 se convierte en un emisor. El mensaje se envía efectivamente cuando el retraso aleatorio (definido por RAD) termina. La función umbral, que permite a DFCN facilitar el mensaje de re-difusión cuando la conectividad es baja depende del tamaño del vecindario s, como se muestra en:

$$\text{threshold}(n) = \begin{cases} 1 & \text{si } n \leq \text{safeDensity} \\ \text{minGain} * n & \text{en otro caso} \end{cases},$$

donde *safeDensity* es la densidad de seguridad máxima por debajo de la cual DFCN siempre continua con el proceso de difusión y *minGain* es un parámetro de DFCN utilizado para calcular el umbral mínimo para continuar con el envío de un mensaje, es decir, la proporción entre el número de vecinos que no han recibido el mensaje y el número total de vecinos.

Cada vez que un dispositivo s descubre un nuevo vecino, el RAD de este dispositivo para todos los mensajes se pone a cero y, por tanto, los mensajes son inmediatamente candidatos al envío. Si $N(s)$ es mayor que un umbral dado, llamado *proD*, este comportamiento se desactiva, por lo que no se podrá realizar ninguna acción sobre el nuevo vecino descubierto.

15.2.3. Definición de MOPs

En esta sección definimos dos problemas multi-objetivo, que se presentan por primera vez en este trabajo, para la optimización del protocolo de difusión en MANETs. De la descripción de DFCN proporcionada en la sección anterior, los siguientes parámetros deben ser ajustados:

MinGain es la ganancia mínima para continuar el proceso de difusión. Este es el parámetro más importante para ajustar DFCN, ya que minimizar el ancho de banda debe depender mucho de la densidad de la red. Los valores que puede tomar este parámetro varían entre 0.0 y 1.0.

[límiteInferiorDeRAD, límiteSuperiorDeRAD] definen el intervalo de valores que puede tomar RAD (retraso aleatorio para re-difusión en milisegundos). Los dos parámetros toman valores del intervalo $[0.0, 10.0]$ en milisegundos.

proD es la densidad máxima ($proD \in [0, 100]$) para la que todavía se necesita usar el comportamiento proactivo (reactivo en los nuevos vecinos) para complementar el comportamiento reactivo.

safeDensity define una densidad máxima de seguridad del umbral "threshold(n)", que varía entre 0 y 100 dispositivos.

Estos cinco parámetros, es decir, cinco variables de decisión que corresponden a una configuración de DFCN, caracterizan el espacio de búsqueda de nuestro MOP. Hemos establecido intervalos suficientemente amplios para los valores de estos parámetros ya que queremos incluir todas las posibilidades razonables que podamos encontrar en el escenario real. Nótese que algunos de los parámetros utilizados son números enteros, mientras que otros toman valores reales, por lo que el algoritmo de optimización que trate este problema debe trabajar en los dominios discreto y continuo a la vez. Los objetivos a optimizar son (ver Ecuación 15.1): minimizar la duración del proceso de difusión, maximizar la cobertura de la red, y minimizar el número de transmisiones. Por eso, hemos definido un MOP de tres objetivos, denominado DFCNT (ajuste de DFCN). Como constatamos antes, este problema se define para una red dada en la que se usa la estrategia de difusión de DFCN. Ya que se han considerado tres MANETs metropolitanas distintas del mundo real, se resuelven tres instancias de DFCNT: *DFCNT.CentroComercial*, *DFCNT.Metropolitano*, y *DFCNT.Autopista*.

$$DFCNT \begin{cases} f_1(x) = \text{duración del proceso de difusión} & -\text{minimizar} \\ f_2(x) = \text{cobertura} & -\text{maximizar} \\ f_3(x) = \text{número de transmisiones} & -\text{minimizar} \end{cases} \tag{15.1}$$

Además, también es interesante considerar la cobertura de la red como una restricción en vez de como un objetivo para propósitos prácticos. De esta forma, podemos definir que la cobertura debe estar, por ejemplo, por encima del 90 %, y entonces encontrar la mejor relación entre el ancho de banda y la duración. El problema resultante, que se llama cDFCNT (DFCNT*con restricción*), es un MOP bi-objetivo con una restricción. Análogamente, tres redes diferentes aquí traen tres instancias distintas de cDFCNT (ver Ecuación 15.2): *cDFCNT.CentroComercial*, *cDFCNT.Metropolitano*, y *DFCNT.Autopista*.

$$cDFCNT \begin{cases} f_1(x) = \text{duración del proceso de difusión} & -\text{minimizar} \\ f_2(x) = \text{número de transmisiones} & -\text{minimizar} \\ g(x) \ = \text{cobertura} \geq 90\,\% \end{cases} \tag{15.2}$$

15.3. Un cGA Multi-Objetivo: cMOGA

En esta sección presentamos cMOGA, un algoritmo multi-objetivo basado en un modelo canónico de cGA. Como se advirtió en el Capítulo 11, aunque existen otras aproximaciones algorítmicas previas basadas en un modelo celular en la literatura, ninguna de ellas, sigue el modelo canónico de cGA. En el Algoritmo 15.1, damos el pseudocódigo de cMOGA. Podemos observar que el Algoritmo 15.1 es muy parecido al modelo canónico de cGA presentado en el Capítulo 1. Una de las diferencias principales entre los dos algoritmos es la existencia de un *frente de Pareto* (Definición 11.5) en el caso del cGA multi-objetivo. El frente de Pareto no es más que una población adicional compuesta por las soluciones no dominadas encontradas hasta el momento, y que tiene un tamaño máximo. Con la idea de poder administrar las soluciones en el frente de Pareto para conseguir obtener un conjunto diverso, utilizamos el proceso de *crowding*.

Algoritmo 15.1 Pseudocódigo de cMOGA.

```
1.  proc Evoluciona(cmoga)        //Parámetros del algoritmo en 'cmoga'
2.  Frente_Pareto = CrearFrente() //Crea un Frente de Pareto vacío
3.  mientras !CondiciónParada() hacer
4.    para individuo ← 1 hasta cmoga.TamañoPobl hacer
5.      vecindario ← CalculaVecindario(cmoga,posición(individuo));
6.      padres ← Selección(vecindario);
7.      descendiente ← Recombinación(cmoga.Pc,padres);
8.      descendiente ← Mutación(cmoga.Pm,descendiente);
9.      Evaluación(descendiente);
10.     Reemplazo(posición(individuo),pobl_auxiliar,descendiente);
11.     InsertarFrentePareto(individuo);
12.   fin para
13.   cmoga.pop ← pobl_auxiliar;
14. fin mientras
15. fin proc Evoluciona;
```

cMOGA comienza creando un frente de Pareto vacío (línea 2 del Algoritmo 2). Los individuos se sitúan en una rejilla toroidal de 2 dimensiones y se le aplican sucesivamente los operadores genéticos (líneas 7 y 8) hasta que llegamos a la condición de parada (línea 3). Para cada individuo el algoritmo selecciona dos padres de su vecindario, los recombina para obtener un descendiente, lo muta, evalúa al individuo resultante y lo inserta si el individuo actual no lo domina tanto en la población auxiliar como en el frente de Pareto (siguiendo el mismo proceso de *crodwing* utilizado en [80]) –líneas 10 a 13. Finalmente, después de cada generación, la población auxiliar reemplaza a la población antigua.

15.3.1. Tratando con Restricciones

Para afrontar MOPs con restricciones, cMOGA utiliza una aproximación simple que también podemos encontrar en otros algoritmos evolutivos multi-objetivo, como NSGA-II [80] y microGA [59]. En dicha aproximación, siempre que comparamos dos individuos, se comprueban sus restricciones. Si los dos son factibles, se les aplica directamente un análisis de dominancia de Pareto (Definición 11.3). Si uno es factible y el otro no, el primero domina. En otro caso, si ninguno de los dos son factibles, el que tenga la menor cantidad de violaciones de restricciones dominará al otro.

15.4. Experimentos

En esta sección, vamos a describir primero la parametrización que utiliza cMOGA. Después, analizaremos las configuraciones del simulador de redes MANET para los tres entornos definidos. Finalmente, compararemos los resultados obtenidos para *DFCNT* y *cDFCNT*.

cMOGA se ha implementado en Java (pertenece a la biblioteca JCell) y ha sido probado en un PC a 2.8 GHz con un procesador Pentium IV con 512 MB de memoria RAM y ejecutando Suse Linux 8.1 (kernel 2.4.19-4GB). La versión de Java utilizada es 1.5.0_05.

Tabla 15.1: Parametrización utilizada en cMOGA.

Tamaño de la Población	100 Individuos (10×10)
Condición de Parada	25000 Evaluaciones de Función
Vecindario	NEWS
Selección de los Padres	Individuo Actual + Torneo Binario
Recombinación	Binario Simulado, $p_c = 1.0$
Mutación	Polinomio, $p_m = 1.0/L$ (L = Longitud del Individuo)
Reemplazo	Reemp_si_No_Domina
Tamaño del Archivo	100 Individuos
Proceso de Crowding	Rejilla Adaptativa

15.4.1. Parametrización de cMOGA

En la Tabla 15.1 mostramos los parámetros que utiliza cMOGA. Para estructurar la población hemos escogido una rejilla toroidal cuadrada de 100 individuos. El vecindario utilizado se compone de 5 individuos: el considerado más los que se encuentran situados al norte, sur, este y oeste (vecindario NEWS). Debido a la naturaleza estocástica del simulador Madhoc, se realizan cinco simulaciones por cada función de evaluación y se calcula el valor de fitness como la media de los valores obtenidos en cada una de estas simulaciones. Esto tiene una influencia considerable en el tiempo de ejecución para resolver el problema, y explica por qué obtener 30 ejecuciones independientes en las pruebas de nuestro algoritmo representa un gran esfuerzo al estudiar el problema, ya que queremos asegurar que haya significancia estadística en los resultados.

Utilizamos el operador de *recombinación binaria simulada* (SBX) [79] con probabilidad $p_c = 1.0$. Tal y como su nombre sugiere, SBX simula el principio de funcionamiento de una recombinación de un sólo punto en genotipos binarios. El operador de mutación utilizado es la *mutación polinómica* [79], y se aplica a cada alelo de los individuos con probabilidad $p_m = 1.0/L$. El descendiente resultante reemplaza al individuo en la posición actual si éste último no domina al primero. Para insertar individuos en el frente de Pareto, se utiliza un *algoritmo con rejilla adaptativa* [168]. Consiste en dividir el espacio objetivo en hipercubos con el objeto de equilibrar la densidad de las soluciones no dominadas que hay en los hipercubos. Entonces, cuando se inserta una solución no dominada en el frente de Pareto, se determina su localización en la rejilla del espacio de soluciones. Si el frente de Pareto ya está lleno (contiene 100 soluciones) y la localización en la rejilla de la nueva solución no está contenida en el hipercubo más poblado, se elimina una solución que pertenezca a este hipercubo antes de insertar la nueva.

15.4.2. Configuración de Madhoc

Como afirmamos en la Sección 15.2.1, hemos definido tres entornos distintos para las MANETs simulando tres posibles escenarios que se pueden encontrar en el mundo real. Sus características principales se explican en esta sección, y se resumen en la Tabla 15.2. Mostramos en la Figura 15.3 un ejemplo de MANET para cada uno de los escenarios estudiados. Estos ejemplos de red se obtuvieron utilizando el interfaz gráfico de MadHoc con la parametrización que se sugirió en nuestro conjunto de problemas propuestos. Consideramos que la difusión se completa cuando la cobertura alcanza el 100 % o cuando no varía en un periodo de tiempo razonable (establecido en 1.5 segundos tras realizar algunos experimentos). Este punto es importante ya que una condición de parada impropia nos conduciría hacia malos resultados o simulaciones lentas.

Tabla 15.2: Características principales de los entornos propuestos.

		Centro Comercial	**Metropolitano**	**Autopista**
Superficie		40 000 m^2	160 000 m^2	1 000 000 m^2
Densidad de círculos		800 (tiendas/km^2)	50 (cruces/km^2)	3 (uniones/km^2)
Dispositivos	Velocidad fuera de los círculos	0.3–1 m/s	1–25 m/s	30–50 m/s
	Velocidad dentro de los círculos	0.3–0.8 m/s	0.3–10 m/s	20–30 m/s
	Densidad	2 000 dev./km^2	500 dev./km^2	50 dev./km^2
Obstrucción de muros		70 %	90 %	0 %

El Entorno de Centro Comercial

En esta sección procedemos a explicar la parametrización que utilizamos para modelar el entorno del centro comercial. En los centros comerciales, la densidad tanto de las tiendas (círculos) como de los dispositivos es normalmente muy grande. También, existen paredes que atenúan la señal y limitan los movimientos de los dispositivos, que además suelen ser muy lentos, ya que simulan personas que caminan. Hemos definido para este trabajo un centro comercial de 200×200 metros cuadrados de superficie con densidad de 800 tiendas y 2000 dispositivos por kilómetro cuadrado. Las tiendas son círculos cuyo radio oscila entre 1 y 10 metros, y la obstrucción de las paredes se calcula como una penalización de la fuerza de la señal del 70 %. Finalmente, los dispositivos viajan con una velocidad entre 0.3 y 1 m/s en los pasillos y entre 0.3 y 0.8 m/s cuando están dentro de las tiendas.

Con respecto al entorno del centro comercial, merece la pena destacar que el grafo de conexión resultante es bastante denso (véase la Figura 15.3). La razón por la que sucede esto es porque la cobertura de los dispositivos móviles oscila entre 40 y 80 metros (valor elegido aleatoriamente), y el área de simulación es pequeño. Por tanto, los problemas *DFCNT.CentroComercial* y *cDFCNT.CentroComercial* son más difíciles de resolver debido al problema de la tormenta en la difusión [222]. Este problema consiste en la aparición de congestiones intensas en la red provocadas por el reenvío de paquetes debido a la alta frecuencia de colisión de paquetes.

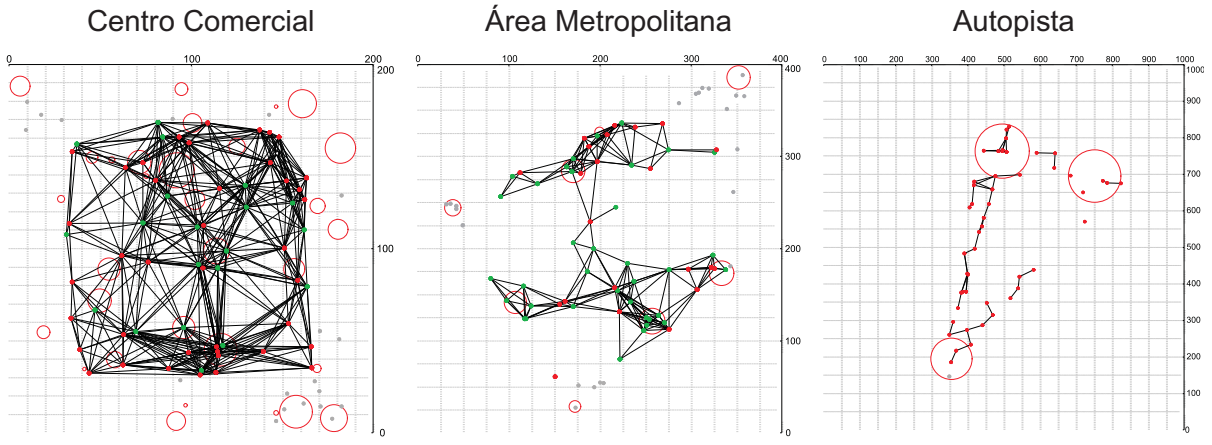


Figura 15.3: Los tres escenarios estudiados para las MANETs.

El Entorno Metropolitano

En este segundo entorno estudiamos el comportamiento de DFCN en una MANET metropolitana. Para simular este entorno establecemos una superficie de 400×400 metros cuadrados, con una densidad de 50 círculos por kilómetro cuadrado (representando cruces entre las calles) con una superficie circular cuyo radio varía entre 3 y 15 metros. La obstrucción de las paredes en este caso es mayor que en el entorno del centro comercial (llega hasta el 90 %), y la densidad de los dispositivos es de 500 elementos por kilómetro cuadrado. Cuando establecimos la velocidad de los dispositivos, tuvimos en cuenta los distintos casos entre personas caminando o viajando en coche, por lo que su valor oscila entre 0.3 y 10 m/s cuando están en un cruce y entre 1 y 25 m/s en el resto del escenario (cuando están por las calles).

En la Figura 15.3, se puede observar que la red resultante en un área metropolitana no es tan densa como en el entorno del centro comercial. Comúnmente hablando, esta clase de redes se componen de unas pocas subredes, que se conectan normalmente unas a otras con unos pocos enlaces, normalmente uno o dos, o incluso cero (subredes no conectadas). Además, puede que algunos dispositivos no formen parte de ninguna subred (nodos aislados). La topología de esta red puede cambiar muy rápidamente, ya que algunos de los dispositivos viajan a altas velocidades dentro de los vehículos. Todas estas características dificultan la tarea de difusión y hace que para nosotros sea interesante el estudio de esta clase de redes.

El Entorno de la Autopista

Como ya comentamos en la Sección 15.2.1, este entorno se compone por unos pocos dispositivos, que viajan a muy altas velocidades. Por tanto, utilizamos el escenario *entorno humano* de MadHoc para simular la red con la peculiaridad de establecer una obstrucción de pared del 0 %. La superficie simulada es de 1000×1000 metros cuadrados con una densidad de sólo 50 dispositivos por kilómetro cuadrado. Estos dispositivos viajan a velocidades aleatorias comprendidas entre 30 y 50 m/s. Definimos las carreteras como líneas rectas que conectan dos círculos, y establecemos una densidad de sólo tres círculos (entradas de la autopista y/o salidas). El tamaño de cada círculo (longitud de la entrada/salida) se establece en un valor aleatorio que varía entre 50 y 200 metros (radio de los círculos $\in [25, 100]$ metros).

Se puede ver en la Figura 15.3 cómo la red resultante utilizando esta parametrización está compuesta por un conjunto de múltiples subredes (usualmente inconexas) involucrando un número pequeño de dispositivos (incluso sólo uno). La existencia de estas redes pequeñas y aisladas supone un duro obstáculo para la tarea del protocolo de difusión. Además, la topología de la red cambia muy rápidamente debido a las altas velocidades en el movimiento de los dispositivos. Por tanto, como consecuencia de estas altas velocidades, están continuamente creándose y desapareciendo redes, lo que dificulta aún más el proceso de difusión.

15.4.3. Resultados para DFCNT

Ahora presentamos y analizamos los resultados obtenidos para el problema *DFCNT* (Sección 15.2.3) con los tres entornos. Vamos a recordar que este problema se compone de cinco variables de decisión y tres funciones objetivo. Todos los valores que se presentan se promedian sobre 30 ejecuciones independientes de cMOGA.

En la Tabla 15.3 mostramos la media y la desviación estándar del tiempo de ejecución (en horas) y el número de soluciones no dominadas que encuentra cMOGA para las tres instancias de *DFCNT*: *DFCNT.CentroComercial*, *DFCNT.Metropolitano* y *DFCNT.Autopista*. Como se puede observar, el tiempo de cómputo de una ejecución individual es muy largo, ya que varía desde 1.98 días para el escenario

Tabla 15.3: Resultados de cMOGA para las tres instancias *DFCNT*.

Entorno	**Tiempo (h)**	**Número de óptimos de Pareto**
DFCNT.CentroComercial	66.12 ± 7.94	97.77± 3.20
DFCNT.Metropolitano	108.21± 8.41	93.40±18.02
DFCNT.Autopista	47.57 ± 0.42	52.27±10.99

de la autopista hasta 4.51 días en el caso de *DFCNT.Metropolitano*. La razón es el alto coste de calcular la función de fitness, ya que lanzamos cinco simulaciones por cada evaluación, y cada ejecución del simulador necesita entre 1 y 4 segundos. En lo referente al número de soluciones encontradas, los resultados obtenidos son altamente satisfactorios en las tres instancias *DFCNT*, ya que el número de diferentes soluciones no dominadas encontradas es 97.77, 93.40, y 52.27 en media (el máximo es 100) para *DFCNT.CentroComercial*, *DFCNT.Metropolitano* y *DFCNT.Autopista*, respectivamente. Por tanto, permitimos al diseñador elegir entre un amplio rango de posibilidades. Nótese que el número de soluciones que componen el frente de Pareto disminuye conforme la densidad de dispositivos de la red es menor (aumentando la superficie y disminuyendo el número de dispositivos). Esto es porque empleamos el mismo criterio para considerar que termina el proceso de difusión en tres entornos muy distintos. Como trabajo futuro, planeamos adaptar este criterio a cada entorno, lo que esperamos nos conducirá a obtener mejores resultados.

Como ejemplo de la diversidad y del amplio conjunto de soluciones que ofrece el optimizador multiobjetivo, representamos gráficamente en la Figura 15.4 un ejemplo de un frente de Pareto obtenido con cMOGA para los tres entornos estudiados. Las mejores soluciones son las que implican (i) alta cobertura, (ii) bajo ancho de banda, y (iii) una corta duración del proceso de difusión, siendo los parámetros más importantes (i) y (ii). De hecho, los óptimos de Pareto en estos frentes que alcanzan una cobertura de más del 95 % necesitan en media 3.77 segundos y 17.25 mensajes para el centro comercial, y 13.78 segundos y 75.71 mensajes (intensa utilización del ancho de banda) en el caso de un área metropolitana. Por contra, en el caso del entorno de la autopista sólo cinco soluciones del frente de Pareto tienen más del 95 % de cobertura (38 para *DFCNT.CentroComercial* y 16 para *DFCNT.Metropolitano*), alcanzando una media de 74.88 mensajes enviados en 13.37 segundos, que son valores parecidos a los del área metropolitana. Finalmente, podemos notar que si la cobertura no fuese una gran restricción en nuestra red, podríamos usar soluciones muy económicas en términos de tiempo y ancho de banda para los dos entornos.

Comparando los tres gráficos, es posible observar que la difusión es más eficiente en el entorno del centro comercial que en los otros dos casos, ya que siempre tarda menos de 8 segundos (duración), transmite menos de 23 mensajes (ancho de banda), y alcanza más del 40 % de los dispositivos (cobertura). En los entornos metropolitano y de autopista, el proceso de difusión tarda más, con un mayor número de mensajes transmitidos y la cobertura en algunos casos es menor que el 10 %. Finalmente, aunque el frente obtenido para el entorno de la autopista tiene valores límite similares a los que se observan en el área metropolitana, la diversidad es mucho menor en el escenario de la autopista.

La diferencia de la cobertura entre los conjuntos de soluciones no dominadas encontrados en los tres entornos es un resultado esperable de forma intuitiva, ya que la probabilidad de tener subredes aisladas (compuestas por uno o más dispositivos) crece conforme aumenta la superficie de simulación y disminuye el número de dispositivos.

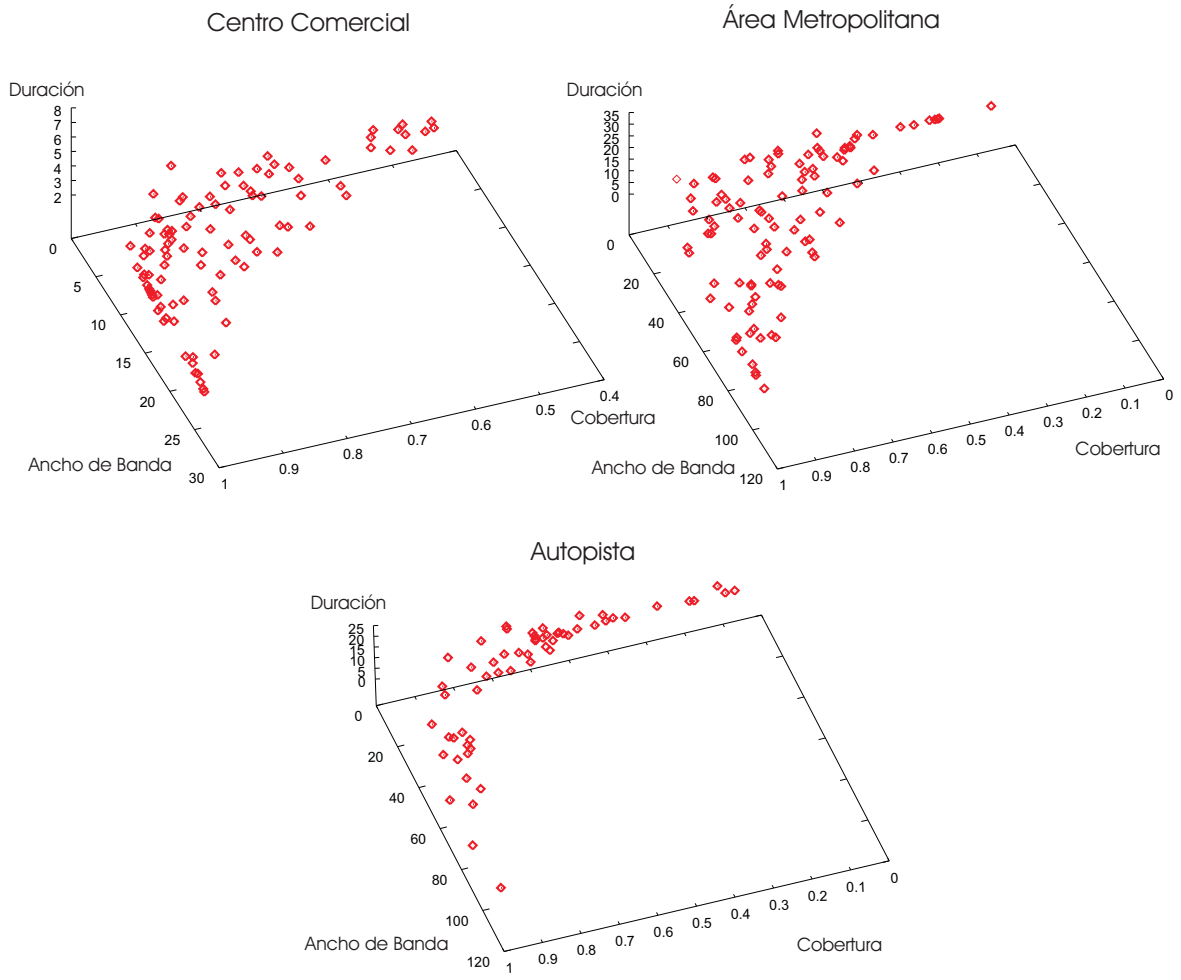


Figura 15.4: Frentes de Pareto para los tres entornos y el problema *DFCNT*.

Por tanto, la diferencia en la calidad de las soluciones es consecuencia de las diferentes topologías de las redes, ya que el alto grado de conectividad de los dispositivos en el entorno del centro comercial permite que un mensaje llegue a muchos más dispositivos que en el caso de los otros dos entornos estudiados. Contrariamente, este alto grado de conectividad aumenta el riesgo de aparición del problema de la tormenta de difusión, haciendo que *DFCNT.CentroComercial* sea muy difícil de resolver. A partir de nuestros resultados podemos concluir que cMOGA ha tratado el problema con éxito.

Los frentes de Pareto de la Figura 15.4 completan los objetivos de diseño del protocolo de DFCN: la mayoría de los puntos (en el centro de las nubes) se corresponden con conjuntos de parámetros que hacen que DFCN alcance una cobertura cercana al 100 %, manteniendo muy baja la utilización de la red. Lo que hace que el problema DFCNT sea particularmente interesante desde el punto de vista de la aplicación, es que permite al diseñador descartar este comportamiento por defecto estableciendo un *grado*

Tabla 15.4: Resultados de cMOGA para las tres instancias *cDFCNT*.

Entorno	Tiempo (h)	Número de óptimos de Pareto
cDFCNT.CentroComercial	56.53±10.56	13.47±2.70
cDFCNT.Metropolitano	106.15± 9.11	5.57±1.98
cDFCNT.Autopista	46.99± 4.32	3.40±1.76

de cobertura para la aplicación de difusión. De hecho, no todas las aplicaciones necesitan maximizar la tasa de cobertura. Por ejemplo, un *anuncio local* –que consiste en dispersar mensajes publicitarios a dispositivos que estén alejados unos pocos saltos de la fuente– necesita que el proceso de difusión cese después de un rato. A veces también se quiere evitar una alta cobertura. Por ejemplo, intentar conseguir una alta cobertura en una MANET metropolitana (que realmente puede estar formada por miles de dispositivos) es perjudicial, ya que esto nos llevaría a intensas congestiones en la red.

15.4.4. Resultados para cDFCNT

Ahora analizaremos los resultados del problema *cDFCNT*, donde la cobertura se ha convertido en una restricción: al menos el 90 % de los dispositivos deben recibir el mensaje de difusión. De esta manera, cMOGA tiene que encontrar soluciones con una relación entre el ancho de banda y la duración del proceso de difusión. Como en el caso de *DFCNT*, la Tabla 15.4 presenta el tiempo medio y el número de óptimos de Pareto que cMOGA es capaz de encontrar en 30 ejecuciones independientes al resolver *cDFCNT* para las tres instancias: *cDFCNT.CentroComercial*, *cDFCNT.Metropolitano*, y *cDFCNT.Autopista*.

En lo referente al tiempo de ejecución, *cDFCNT* se puede resolver un poco más rápido que su equivalente sin restricciones para el entorno del centro comercial (56.53 frente a 66.12 horas), aunque la diferencia es menor (de alrededor del 2 %) en los casos del área metropolitana (106.15 frente a 108.21 horas) y del entorno de la autopista (46.99 frente a 47.57).

Dos razones pueden explicar este comportamiento. Primero, los análisis de dominancia son menos costosos debido a que tienen menos objetivos, reduciendo así el tiempo necesario para comprobar si una nueva solución es dominada o no por el frente actual. Segundo, al encontrar un menor número de puntos para el MOP con restricciones, el análisis anterior es todavía menos costoso debido al bajo número de comparaciones necesarias. Como se afirmó anteriormente, si analizamos las soluciones no dominadas encontradas, los resultados muestran que el número de puntos ha sido drásticamente reducido en los tres entornos estudiados respecto a las instancias sin restricciones. Esto indica que la restricción de la cobertura hace que el problema sea muy difícil de resolver (especialmente para los entornos metropolitano y de autopista), quizás excesivamente duro si las soluciones propuestas en *DFCNT* satisfacen al diseñador.

La Figura 15.5 representa tres frentes de Pareto típicos que se corresponden con las tres instancias propuestas de *cDFCNT*. La relación entre las dos funciones objetivo está clara en los tres casos: si un mensaje tiene que ser rápidamente difundido, implica utilizar un gran ancho de banda. Por otro lado, se podrían obtener soluciones baratas en términos de ancho de banda considerando largos tiempos de duración. Una vez más, puede verse que las soluciones para los entornos metropolitano y de autopista son más caras que en el caso del entorno del centro comercial (en lo que se refiere a tiempo y ancho de banda). El bajo número de puntos encontrados para los entornos metropolitano y de autopista con respecto al entorno del centro comercial se puede explicar por la baja cobertura final que se encuentra en las soluciones debido a la topología de la red, como ya comentamos en el caso del problema *DFCNT*.

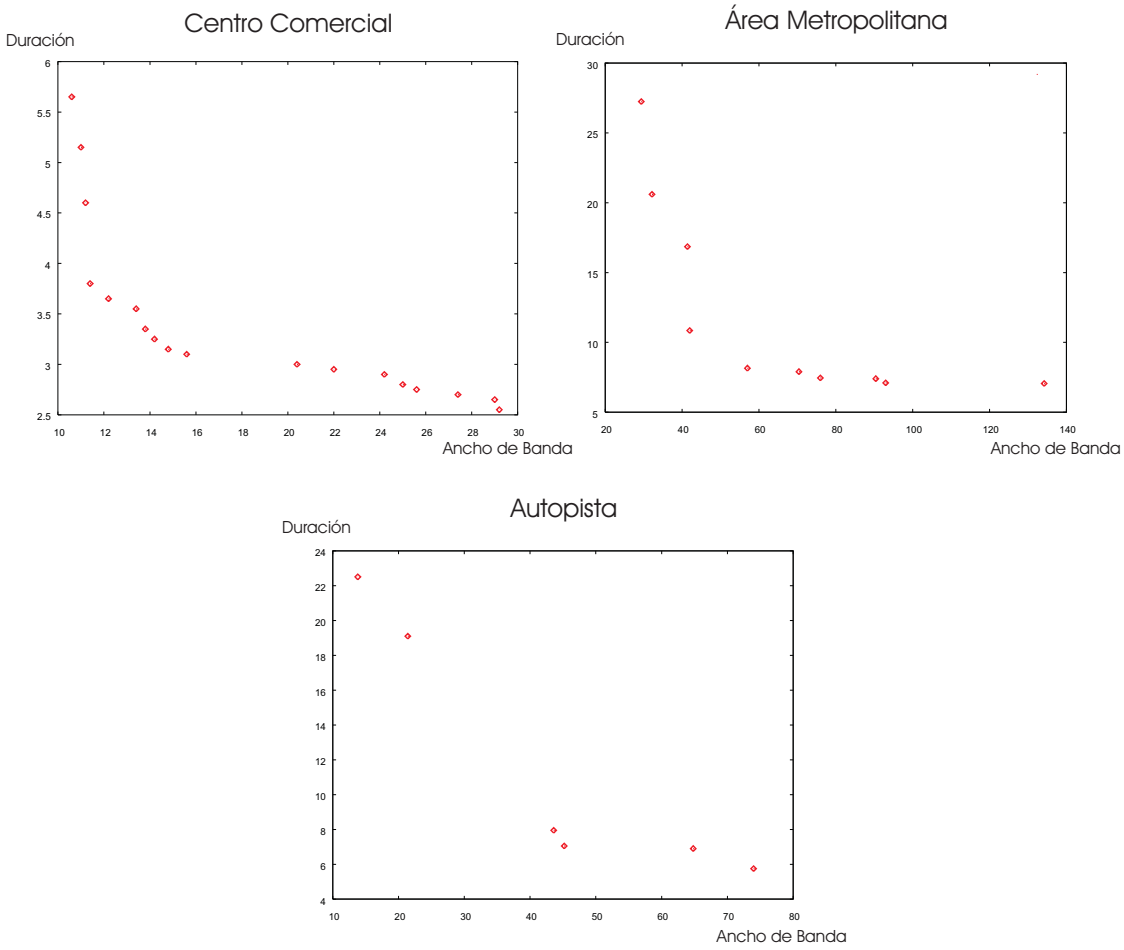


Figura 15.5: Frente de Pareto para los tres entornos y el problema *cDFCNT*.

Como sugerimos antes, todos los protocolos de difusión siguen esta regla: cuanto más oportunistas, más rápidos son (sin considerar el impacto de las colisiones de los paquetes), pero más ancho de banda utilizan. DFCN ha sido diseñado teniendo este aspecto en cuenta: su comportamiento –cuando se usa con los parámetros adecuados– es una excepción a esta regla. De todas maneras, tenemos distintos objetivos en este estudio, ya que estamos buscando el conjunto óptimo de Pareto de los parámetros. Consecuentemente, se muestra en los frentes de Pareto un diverso subconjunto de comportamientos exhibidos por todos los protocolos de difusión. En estos frentes de Pareto podemos observar que conseguir tiempos de propagación muy cortos conlleva un gran ancho de banda y que sólo podemos conseguir un ancho de banda bajo utilizando políticas con pocos reenvíos de paquetes. Aparte de este comportamiento asintótico, los frentes de Pareto también muestran que DFCN puede ser ajustado de tal manera que permita obtener una duración razonablemente corta del proceso de difusión mientras que mantiene una baja ocupación de la red (por ejemplo, número de envíos de paquetes). Al tener garantizada una cobertura elevada ($\geq 90\,\%$), estas parametrizaciones son apropiadas para la mayoría de las aplicaciones de difusión.

Tabla 15.5: Comparación de resultados entre *DFCNT* y *cDFCNT*.

		DFCNT	**cDFCNT**
Centro Comercial	Tiempo (horas)	66.12± 7.94	56.53±10.56
	Óptimos de Pareto	97.77± 3.20	93.40±18.02
Metropolitano	Tiempo (horas)	108.21± 8.41	106.15± 9.11
	Óptimos de Pareto	93.40±18.02	5.57± 1.98
Autopista	Tiempo (horas)	47.57± 0.42	46.99± 4.32
	Óptimos de Pareto	52.27±10.99	3.40± 1.76

15.4.5. Comparando DFCNT y cDFCNT

En esta sección, comparamos la complejidad del algoritmo propuesto sin restricciones con respecto a su versión con restricciones. Para ello, mostramos un resumen en la Tabla 15.5 de los resultados mostrados en las tablas 15.3 y 15.4. Como afirmamos anteriormente, se pueden deducir dos hechos de estos resultados: primero, resolver las instancias de *DFCNT* es más costoso que resolver las de *cDFCNT* en términos de esfuerzo computacional (la diferencia se hace más importante conforme aumenta el número de dispositivos) y, segundo, cMOGA es capaz de encontrar un mayor número de óptimos de Pareto para *DFCNT* que para *cDFCNT*.

Para proporcionar relevancia desde el punto de vista estadístico, hemos realizado varios análisis estadísticos (t-tests) con el 95 % del nivel de significancia. Estos t-tests se han realizado después de asegurar que los datos siguen una distribución normal (utilizando el análisis de Kolmogorov-Smirnov). Los análisis muestran diferencias significativas para todos los resultados de la Tabla 15.5, excepto cuando comparamos las instancias metropolitana y de autopista en términos del tiempo de ejecución. Por eso, si prestamos atención al número medio de soluciones encontradas en los frentes de Pareto, cMOGA tiene mayor dificultad para resolver *cDFCNT* que para *DFCNT*, con relevancia estadística en todos los casos estudiados. De este modo, podemos concluir que *cDFCNT* es un problema más complejo ya que, aunque los problemas sin restricciones (*DFCNT*) necesitan un mayor esfuerzo computacional (tiempos de ejecución más largos) que los que tienen restricciones, sólo encontramos diferencias estadísticamente significativas en el caso del entorno del centro comercial.

15.5. Comparación de cMOGA con Otras Metaheurísticas

En esta sección pasamos a realizar un estudio comparativo de cMOGA con otras metaheurísticas para el problema DFCNT. Se ha seleccionado para este estudio únicamente el entorno del centro comercial, que es quizás el más complicado de los tres entornos propuestos debido al problema de la tormenta de la difusión. Comparamos cMOGA con NSGA-II, un algoritmo del estado del arte en optimización multi-objetivo, y AbYSS, un algoritmo multi-objetivo de búsqueda dispersa recientemente propuesto cuyos detalles de implementación pueden consultarse en [187, 219].

15.5.1. Parametrización

En esta sección presentamos los parámetros de los algoritmos estudiados. Los tres algoritmos se detienen cuando se han calculado 25000 funciones de evaluación de DFCNT. Todos ellos tienen un tamaño se archivo de 100 soluciones no dominadas. Las configuraciones de SBX (operador de recombinación en NSGA-II y cMOGA, y método de *combinación de solución* en AbYSS) y de la mutación polinómica

Tabla 15.6: Número de óptimos de Pareto e hipervolumen para NSGA-II, cMOGA, y AbYSS al resolver DFCNT.

Métrica	**Medida**	**NSGA-II**	**cMOGA**	**AbYSS**	**Test**
Número de Óptimos de Pareto	$\tilde{x}$	100	99	100	+
	IQR	0	5	1	
	max	100	100	100	
	min	95	88	89	
Hipervolumen	$\tilde{x}$	0.8679	0.8392	0.8625	+
	IRQ	0.0067	0.0574	0.0404	
	max	0.8749	0.8616	0.8728	
	min	0.8529	0.7714	0.7463	

Tabla 15.7: Valores de la cobertura de conjuntos para NSGA-II, cMOGA, y AbYSS al resolver DFCNT.

$\mathcal{C}(\mathbf{A},\mathbf{B})$		**Medidas**				**Test**
A	**B**	$\tilde{\mathbf{x}}$	**IRQ**	**max**	**min**	
NSGA-II	cMOGA	0.6715	0.0438	0.7739	0.5996	+
cMOGA	NSGA-II	0.1937	0.1935	0.2936	0.0148	
NSGA-II	AbYSS	0.5595	0.0612	0.6905	0.4910	+
AbYSS	NSGA-II	0.2714	0.1375	0.3900	0.0214	
cMOGA	AbYSS	0.3834	0.2735	0.5052	0.0987	+
AbYSS	cMOGA	0.5516	0.1380	0.6780	0.0971	

(operador de mutación en NSGA-II y cMOGA y el método de *mejora* en AbYSS) utilizan un índice de distribución de 10. En la parametrización específica de NSGA-II, las tasas de recombinación y mutación se han establecido en 0.9 y 0.2, respectivamente (como sugirieron los creadores). cMOGA se configura con una población de 100 individuos colocados en una rejilla toroidal cuadrada de 10×10 con un vecindario NEWS y una tasa de mutación de 0.2. Finalmente, en AbYSS, el tamaño del conjunto inicial P es 20, el número de $RefSet_1$ y de $RefSet_2$ es 10 soluciones. Es importante notar que 25000 evaluaciones × 5 simulaciones/evaluación significa que DFCN se ha optimizado en más de 125000 instancias de red distintas para cada algoritmo.

15.5.2. Discusión

La comparación de los tres algoritmos se lleva a cabo en esta sección. Se han realizado 30 ejecuciones independientes de cada experimento, y para realizar dichas comparaciones se han utilizado las métricas del *hipervolumen*, el *cubrimiento de conjuntos*, y el número de soluciones no dominadas en el conjunto de soluciones de Pareto encontrado por el algoritmo (ver Capítulo 11).

Los resultados obtenidos aparecen en las tablas 15.6 y 15.7, e incluyen la mediana, $\tilde{x}$, y el rango intercuartílico, IQR, como medida de localización (o tendencia central) y dispersión estadística, respectivamente. Los valores máximos y mínimos de cada medida (max y min) también se presentan. El mismo análisis estadístico realizado en la Sección 15.4 se ha aplicado aquí para que obtener relevancia estadística en nuestras conclusiones.

Si consideramos que los tres algoritmos comparados están configurados para obtener 100 soluciones no dominadas como mucho (máximo tamaño del archivo), destaca en la Tabla 15.6 que la mayoría de los algoritmos comparados llenan el archivo entero de soluciones no dominadas en la mayoría de las ejecuciones. Aunque NSGA-II y AbYSS encuentran un número levemente mayor de óptimos de Pareto en media que cMOGA, las diferencias no son significativas, de forma que los tres algoritmos muestran una capacidad similar de explorar el espacio de búsqueda definido por DFCNT.

La última fila de la Tabla 15.6 presenta los resultados de la métrica *hipervolumen*. Estos resultados muestran que NSGA-II y AbYSS alcanzan valores similares (0.8679 y 0.8625, respectivamente) cuando consideramos la convergencia y la diversidad a la vez en los frentes de Pareto resultantes, y los dos superan a cMOGA en esta métrica (todo esto se sustenta con relevancia estadística).

Los resultados de la *cobertura de conjuntos* se presentan separadamente en la Tabla 15.7 porque suponen una medida no simétrica, y por tanto tienen que calcularse los valores resultantes para todos los pares de combinaciones de los algoritmos. Para cada una de estas combinaciones, el resultado del análisis estadístico explicado anteriormente también se incluye en la última columna. El primer par de comparaciones considera a NSGA-II y cMOGA. Como se puede observar, NSGA-II obtiene mayores valores para esta medida que cMOGA y esto se puede demostrar con relevancia estadística (véase el símbolo "+" en la última columna). Este hecho destaca que NSGA-II puede encontrar soluciones que dominan más soluciones de cMOGA que viceversa, mostrando así una mejor convergencia que el último. Los valores máximos y mínimos también refuerzan esta afirmación, ya que mientras NSGA-II cubre al menos casi el 60 % de las soluciones de cMOGA (valor mínimo de 0.5996), las soluciones del algoritmo celular sólo pueden dominar el 29 % de las soluciones de NSGA-II como máximo (el valor máximo de $\mathcal{C}$ es 0.2936). Si comparamos NSGA-II y AbYSS, los resultados son similares al par previamente analizado: un mejor valor de la mediana para NSGA-II. Pero en este caso destaca que las diferencias entre los dos algoritmos se han reducido ($\mathcal{C}(\text{NSGA}-\text{II}, \text{AbYSS}) = 0.5595$ frente a $\mathcal{C}(\text{AbYSS}, \text{NSGA}-\text{II}) = 0.2714$) y AbYSS es capaz de alcanzar un frente de Pareto que cubre casi el 40 % de las soluciones de NSGA-II ($\max \mathcal{C}(\text{AbYSS}, \text{NSGA}-\text{II}) = 0.3900$). Finalmente, la última comparación implica a cMOGA y a AbYSS. Como se podía esperar de los dos análisis previos de esta métrica, la aproximación de búsqueda dispersa obtiene valores mayores para la *cobertura de conjuntos* que cMOGA (con relevancia estadística). Tenemos que destacar los altos valores de IRQ alcanzados aquí, mostrando así la gran variabilidad resultante de los frentes. Resumiendo todos los pares de combinaciones, se puede observar que más del 50 % de las soluciones no dominadas calculadas por NSGA-II dominan a las soluciones de cMOGA y AbYSS. Esto significa que NSGA-II muestra la mejor convergencia hacia el frente de Pareto óptimo de DFCNT.

Como la métrica de *cobertura de conjuntos* muestra que NSGA-II supera al mejor en términos de convergencia, y según la métrica de *hipervolumen* NSGA-II y AbYSS obtienen valores similares, podemos por tanto concluir que AbYSS alcanza este valor de *hipervolumen* debido a su mejor diversidad en el frente de Pareto encontrado. Esto es, el conjunto de soluciones no dominadas calculadas por AbYSS cubre una región mayor del espacio objetivo, lo que es una importante característica para el diseño actual de MANETs. Los valores máximos de la métrica *hipervolumen* también muestran similitudes entre NSGA-II y AbYSS, incluso cuando el valor mínimo alcanzado por el primero es mayor. Finalmente, mostramos en la Figura 15.6 tres frentes de Pareto que demuestran las afirmaciones anteriores. En términos de diversidad, se puede observar que hay soluciones no dominadas tanto de NSGA-II como de AbYSS que alcanzan configuraciones DFCN donde la cobertura de los mensajes es de alrededor el 30 % de los dispositivos, mientras cMOGA no es capaz de obtener soluciones en esta región del espacio de objetivos (permanecen sobre el 40 % de la cobertura). De hecho, esto no sólo ocurre con la cobertura, sino también con el ancho de banda, donde las soluciones que hacen un intenso uso de la red difunden mensajes

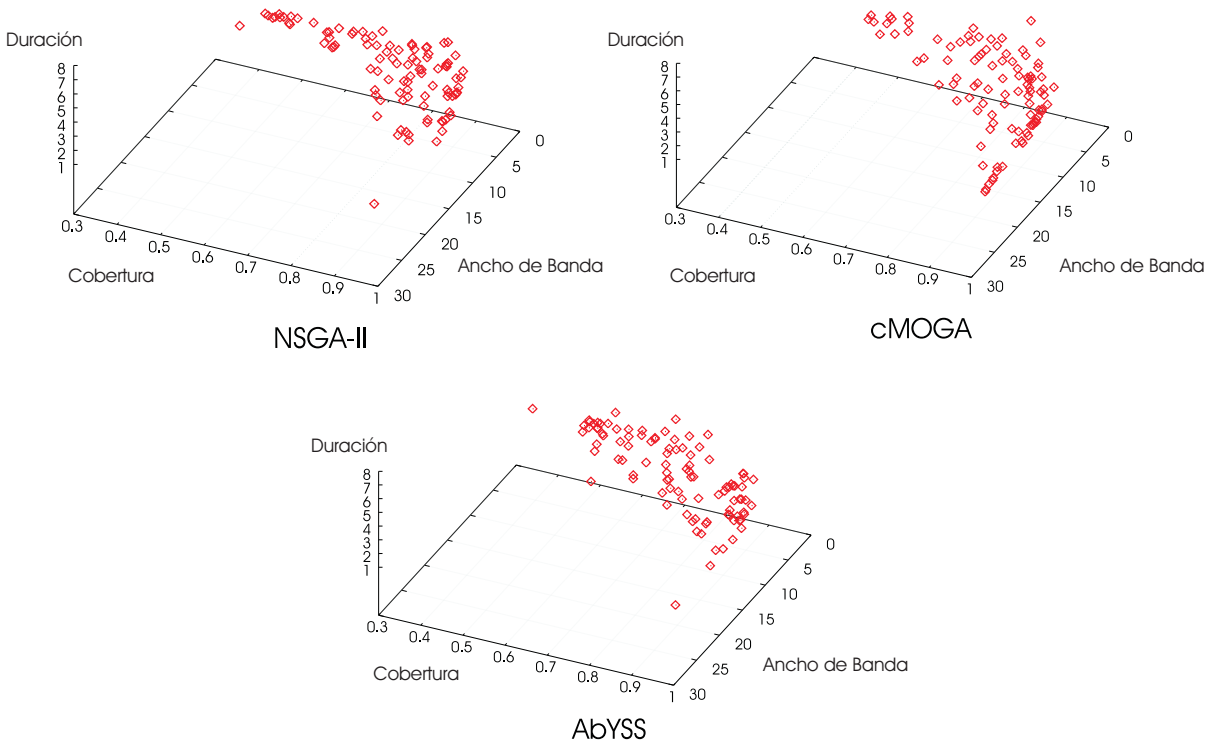


Figura 15.6: Frentes de Pareto de NSGA-II, cMOGA y AbYSS para *DFCNT*.

muy rápidamente. Observe que, de todas formas, AbYSS presenta una levemente mejor dispersión que NSGA-II de las soluciones no dominadas cuando la cobertura oscila entre 0.5 y 0.7 en estos tres frentes de ejemplo. Esto es lo que le permite competir en términos de *hipervolumen* con NSGA-II. Por tanto, utilizar AbYSS proporciona al diseñador de la red (el que toma las decisiones) un conjunto de parametrizaciones más amplio que varía desde configuraciones que alcanzan un alto valor de cobertura en poco tiempo pero utilizan mucho ancho de banda, hasta soluciones baratas en términos de tiempo y ancho de banda que son aptas si la cobertura no es una restricción dura en la red. NSGA-II en cambio proporciona un conjunto de soluciones muy preciso (gran calidad), pero que no están muy bien distribuidas a lo largo del frente de Pareto. Finalmente, cMOGA es el único de los algoritmos que encuentra varias soluciones en los casos de mayor utilización del ancho de banda de la red (más de 10 mensajes enviados). En estos casos, NSGA-II y AbYSS encuentran únicamente una solución que, además, se encuentra aislada del frente en ambos casos, lo que representa una elevada pérdida de diversidad en esa zona del frente para estos dos algoritmos (obsérvese el hueco existente en estos dos frentes en esa zona).

15.6. Conclusiones

Este capítulo presenta una primera aproximación al problema de ajuste óptimo de DFCN, un protocolo de difusión para MANETs, utilizando cMOGA, un nuevo algoritmo celular multi-objetivo. El algoritmo cMOGA se ha utilizado para resolver dos formulaciones del problema: la primera, llamada *DFCNT*, se define como un MOP tri-objetivo, cuyos objetivos son minimizar la duración del proceso de difusión, maximizar la cobertura de la red y minimizar el uso de la red; la segunda, *cDFCNT*, es una versión de dos objetivos de *DFCNT* en la que se considera la cobertura como una restricción en lugar de ser un objetivo.

Se han utilizado tres escenarios realistas diferentes. Por tanto se han resuelto tres instancias distintas para cada MOP. Estas instancias se corresponden con un centro comercial (*DFCNT.CentroComercial* y *cDFCNT.CentroComercial*), las calles en una ciudad (*DFCNT.Metropolitano* y *cDFCNT.Metropolitano*), y un área no metropolitana amplia en la que existen varias carreteras (*DFCNT.Autopista* y *cDFCNT.Autopista*). Nuestros experimentos revelan que resolver las instancias *DFCNT* proporciona un frente de Pareto compuesto por más de 50 puntos para *DFCNT.Highway*, y más de 95 puntos (más de 95 configuraciones distintas DFCN) en el caso de los otros dos entornos, todo ello a costa de invertir una considerable cantidad de tiempo. De todas formas, para un diseñador de red, este tiempo puede ser aceptable dependiendo del objetivo del estudio. Si por el contrario el tiempo fuera una cuestión crítica, *cDFCNT* representa una alternativa interesante un poco más rápida, que asegura una pérdida de cobertura menor al 10 %. De todas formas, el problema resuelto admite un frente de Pareto muy reducido (un menor número de opciones de diseño), indicando que la restricción de la cobertura hace que el problema sea muy duro.

En la segunda parte de nuestro estudio comparamos, sobre el problema *DFCNT.CentroComercial*, cMOGA con otros dos algoritmos multi-objetivo: NSGA-II (perteneciente al estado del arte de algoritmos multi-objetivo) y AbYSS, una nueva propuesta al campo de la optimización multi-objetivo muy competitiva con NSGA-II en términos de convergencia, y que lo mejora en términos de diversidad [219]. Hemos utilizado tres métricas para comparar los algoritmos: número de óptimos de Pareto, *cobertura de conjuntos* e *hipervolumen*. El número de soluciones no dominadas que encuentran NSGA-II y AbYSS es levemente mayor en media que para cMOGA, pero las diferencias no son significativas. Con respecto a la métrica de *cobertura de conjuntos*, los resultados muestran que NSGA-II es capaz de calcular soluciones muy precisas que dominan a la mayoría de soluciones de las otras dos aproximaciones. Por tanto, en lo referente a la convergencia, NSGA-II tiene el mejor rendimiento. Finalmente, la métrica *hipervolumen* demuestra que NSGA-II y AbYSS son similares, superando levemente a cMOGA. Como medida tanto de convergencia como de diversidad, y teniendo en cuenta los resultados de la *cobertura de conjuntos*, podemos afirmar que AbYSS ha alcanzado este valor de hipervolumen debido a una gran ventaja en diversidad, es decir, la distribución de las soluciones a lo largo del frente de Pareto. Desde el punto de vista del diseñador, AbYSS permite escoger una configuración de DFCN de un conjunto más amplio de posibilidades. En cambio, NSGA-II proporciona soluciones de gran calidad con una distribución un poco peor de las soluciones no dominadas a lo largo del frente de Pareto.

Parte IV

Conclusiones y Trabajos Futuros

Conclusiones

En esta tesis se realiza un profundo estudio sobre algoritmos genéticos celulares (cGAs), un tipo de algoritmo evolutivo (EA) que utiliza una población estructurada, de forma que los individuos que la forman sólo pueden relacionarse con sus vecinos más próximos. Inicialmente, se presenta una introducción al concepto de optimización, y a las principales (y más modernas) técnicas de optimización existentes en la actualidad. En esta línea, se presta especial atención a los algoritmos evolutivos, que son un tipo de técnicas aproximadas de optimización muy eficientes al tratar con problemas altamente complejos, que incluso podrían ser difícilmente abordables con otras técnicas. Para un algoritmo evolutivo, se ha demostrado en varios estudios [27, 119] que estructurando la población podemos mejorar su comportamiento con respecto a la versión panmíctica equivalente, lo que nos permite obtener una mayor eficiencia y eficacia en la resolución de los problemas abordados.

Antes de comenzar las investigaciones desarrolladas en este trabajo, se ha realizado una amplia exploración del estado del arte en algoritmos evolutivos celulares, que incluye y clasifica algunas de las publicaciones más importantes relacionadas con este campo. Tras este estudio inicial, se ha propuesto un modelo formal para un cGA canónico, lo que nos ha permitido caracterizar este tipo de algoritmos. Sobre este modelo, proponemos entonces diversas aportaciones al estado del arte en cGAs, que tratan de solventar las lagunas y carencias que hemos encontrado en la literatura existente. En resumen, las principales contribuciones que se aportan en este trabajo de investigación se enumeran a continuación:

1. **Estudio teórico del comportamiento de cGAs.** Existen en la actualidad muy pocos estudios que traten de modelar matemáticamente el comportamiento de los algoritmos evolutivos. Entre ellos cabe destacar los trabajos [119, 246, 260] para el caso de los cEAs, y [24] para los dEAs. En el caso de los EAs celulares, destacamos el trabajo presentado por Giacobini et al. [119] como el más interesante de los existentes ya que, además de que es el que muestra una mayor exactitud, en él se considera la posibilidad de utilizar distintas formas de la población y distintos operadores de selección de individuos. En el Capítulo 5, se ha extendido el estudio presentado en [119] a mecanismos de selección proporcionales (ruleta y ordenación lineal), como se sugiere en las líneas de investigación futura en [119], para los que la aproximación realizada por dicho modelo no es del todo precisa, y obtiene elevadas tasas de error para multitud de los cGAs aproximados. Para resolver dichos problemas, presentamos dos nuevos modelos matemáticos que tratan de mejorar el comportamiento de los existentes. Por un lado, se propone un modelo polinómico que se basa en el valor de la ratio entre las formas de la población y el vecindario. Este modelo es realmente preciso, pero tiene los inconvenientes de no ser lo suficientemente intuitivo ni contemplar la posibilidad del uso de otros métodos de selección distintos del torneo binario, para el que fue diseñado. Por otro lado, se propone también un nuevo modelo probabilístico que se ajusta con mucha mayor precisión que el propuesto por Giacobini et al., como se muestra en los estudios del Capítulo 5, validado en la práctica sobre 75 cGAs distintos. Además, este nuevo modelo propuesto cuenta con la posibilidad de utilizar cualquier tamaño y forma de población y cualquier método de selección existente.

2. **Diseño de nuevos cGAs de población adaptativa.** Como ya se ha comentado en multitud de ocasiones a lo largo de este documento, debido a la forma en que se estructura la población en los cGAs, la velocidad de convergencia del algoritmo disminuye considerablemente con respecto a la de otros GAs (estructurados y no estructurados). El efecto perseguido es mantener la diversidad de soluciones en la población por más tiempo pero, como contrapartida, esto implica la necesidad de un mayor número de evaluaciones por parte del algoritmo para encontrar la solución al problema en

cuestión. Para intentar solventar este problema que presentan los cGAs, se ha propuesto una nueva técnica adaptativa que permita al cGA potenciar la exploración o explotación de las zonas más prometedoras del espacio de búsqueda en función de la velocidad de convergencia de la población. Esta técnica se basa simplemente en modificar la forma de la población, lo que, como se vio en el Capítulo 5, influye marcadamente en el comportamiento del algoritmo. Esta técnica cuenta, además, con la ventaja del bajo coste computacional que requiere, ya que otras técnicas adaptativas existentes en la literatura son tan costosas que en la práctica no merece la pena aplicarlas. Todos los cGAs con población adaptativa estudiados en esta tesis mejoran el comportamiento de todos los cGAs equivalentes (con diversos tipos de poblaciones estáticas y dinámicas pre-programadas) con los que han sido comparados.

3. **Diseño de cGAs con población jerárquica.** En la línea del punto anterior, se pretende con esta aportación modificar la relación entre la exploración y explotación que aplica un cGA clásico sobre el espacio de búsqueda para tratar de mejorar su comportamiento. En este caso, se propone establecer una jerarquía sobre los individuos de la población, de manera que los mejores individuos estarán localizados en una misma zona de la población, potenciando así la explotación de las regiones más prometedoras del espacio de búsqueda. Por otro lado, el resto de los individuos está también localizado en diferentes lugares de la población en función de su valor de fitness, de forma que en estas zonas de la población se promueve la exploración, ya que los individuos menos prometedores se relacionan entre sí, lo que nos permite mantener la diversidad por más tiempo. En el estudio realizado en esta tesis, se ha demostrado el buen comportamiento del uso de cGAs con poblaciones jerárquicas, mejorando a un cGA canónico equivalente para un amplio conjunto de problemas de diversas características.

4. **Diseño de un nuevo algoritmo de estimación de distribuciones celular.** Nuestra aportación en este caso se centra en aprovechar las características de los cGAs en el campo de los EDAs. Así, en este trabajo presentamos por primera vez en una tesis doctoral un EDA celular (cEDA), y realizamos un estudio preliminar en el que se prueba que todos los nuevos cEDAs propuestos mejoran el comportamiento del EDA equivalente con población panmíctica. Para ello, se ha analizado el comportamiento de los algoritmos en un conjunto de problemas de diversas características. Además, también se ha llevado a cabo un análisis teórico, mediante el estudio de la presión de selección, en el que se demuestra que los cEDAs tienen una menor presión de selección que el EDA equivalente, por lo que mantienen la diversidad en la población por más tiempo, evitando así la convergencia prematura a óptimos locales.

5. **Diseño de un nuevo algoritmo memético celular.** En esta tesis se presenta, por primera vez en la literatura, el diseño de un algoritmo celular dentro del importante campo de los algoritmos meméticos [211]. Así, hemos diseñado algoritmos celulares altamente especializados con información del problema a resolver. Esta alta especialización puede ser vista también como un modo de regular la relación entre exploración y explotación del algoritmo en el espacio de búsqueda. Con este tipo de algoritmos, se han obtenido importantes resultados que han pasado a formar parte del estado del arte para los problemas de satisfacibilidad (SAT) y el de rutas de vehículos (VRP).

6. **Diseño de nuevos modelos paralelos de cGAs.** También se han desarrollado dos novedosos modelos de cGAs paralelos. Uno de ellos está diseñado para trabajar en redes de área local (LAN), y ha sido comparado con múltiples modelos paralelos de GAs (con poblaciones centralizadas y distribuidas en islas) sobre el problema SAT, mejorándolos a todos. Además, se ha presentado

un modelo distribuido en islas, en cada una de las cuales se ejecuta un cGA, para trabajar sobre computación *grid*. Con este algoritmo, se ha mejorado el estado del arte para muchos de los problemas del conjunto de instancias VLSVRP (*very large scale VRP*, o VRP de escala muy grande) del problema VRP, que está constituido por las instancias más grandes existentes de este difícil problema.

7. **Diseño de un nuevo cGA multi-objetivo.** Otra importante aportación del presente trabajo de tesis doctoral ha sido la adaptación del modelo celular al campo de la optimización multi-objetivo. Hasta la publicación de este estudio no existía, según nuestro conocimiento, ningún modelo ortodoxo de cGA multi-objetivo. Tan sólo habían sido publicados dos algoritmos multi-objetivo que utilizan algunas características del modelo celular [164, 178]. Se ha comparado el nuevo cGA multi-objetivo propuesto con NSGA-II y PAES, dos EAs multi-objetivo del estado del arte, para un conjunto amplio de problemas, mejorándolos en muchos casos. En particular, cabe destacar que la diversidad del frente de soluciones obtenido por nuestro algoritmo es claramente superior a la de los algoritmos comparados, manteniendo además una proximidad similar a la suya respecto al frente óptimo.

8. **Aplicaciones de cGAs.** Se han validado nuestros nuevos modelos propuestos sobre un amplio espectro de problemas, que pertenecen a los campos de optimización combinatoria y numérica. Además, también se han considerado problemas con variables pertenecientes a ambos dominios (discreto y continuo), así como problemas con restricciones. El uso de un conjunto tan amplio de problemas, algunos de los cuales están tomados del mundo real, junto con el estudio estadístico realizado sobre los resultados, nos permite sustentar nuestras conclusiones de forma especialmente rigurosa frente a otros estudios existentes.

9. **Diseño de JCell.** Por último, se ha construido una biblioteca de algoritmos genéticos celulares programada en Java llamada JCell. Con JCell, cualquier investigador interesado podrá reproducir fácilmente la gran mayoría de los experimentos aquí realizados. Además, JCell ofrece la posibilidad a la comunidad científica de profundizar en cualquiera de los estudios llevados a cabo en este trabajo, y de exportar cualquiera de los modelos presentados a otros tipos de algoritmos, e incluso a otros campos de investigación. Por todo esto, consideramos que el desarrollo y la libre disponibilidad de JCell supone una importante contribución a la comunidad científica.

Como resumen, en este trabajo de tesis doctoral se han realizado numerosas contribuciones al dominio de los algoritmos evolutivos celulares, tanto en el ámbito teórico, como en el algorítmico o en el de aplicación. En el campo teórico, se ha presentado el mejor modelo matemático existente de aproximación a la curva de la presión de selección de un cGA. En cuanto a los nuevos modelos algorítmicos, se proponen nuevos cGAs adaptativos, con población jerárquica, y paralelos. Además, se han presentado también nuevas familias de algoritmos, como los EDAs celulares, los algoritmos meméticos celulares, o los cGAs multi-objetivo. Tanto los nuevos modelos algorítmicos como las familias de algoritmos aquí introducidas por primera vez han sido comparados con sus equivalentes algoritmos panmícticos y/o algoritmos pertenecientes al estado del arte sobre un conjunto importante de problemas de diversas características, y las conclusiones obtenidas están sustentadas por estudios estadísticos.

Por último, se han realizado extensos estudios de algunos de los nuevos algoritmos presentados sobre problemas tomados directamente del mundo real, pertenecientes a áreas tan diversas como la logística o las telecomunicaciones. Ponemos así de manifiesto la utilidad de nuestros modelos para afrontar problemas que se presentan en ámbitos profesionales de la academia y la industria.

Conclusions

In this thesis, we have carried out a deep study on cellular genetic algorithms (cGAs), a kind of Evolutionary Algorithm (EA) which uses a structured population in which individuals can only interact with their nearby neighbors. At the beginning, an introduction to the concept of optimization is given, as well as to the main (and modern) currently existing techniques. Here, we pay special attention to EAs, which are very efficient approximate optimization techniques for dealing with highly complex problems, that hardly could be tackled with other approaches. In the case of EAs, several studies [27, 119] have proved that structuring the population allows them to improve their efficiency and efficacy with respect to their panmitic counterpart when solving the considered optimization problems.

Before starting the studies developed in this work, we have performed a wide exploration of the state-of-the-art in cellular evolutionary algorithms, which includes and classifies some of the most important publications related to this field. After this initial phase, a formal model for a canonical cGA has been defined, thus allowing us to characterize this kind of algorithm. Using this model, we have proposed several contributions to the state-of-the-art in cGAs, which try to overcome the gaps and lacks found in the existing literature. In short, the main contributions of this research work are provided next:

1. **Theoretical study on the behavior of cGAs.** Currently, just some few works exist trying to mathematically model the behavior of EAs. Among them, it is worth mentioning the works [119, 246, 260] in the cGA case, and [24] for dEAs. In cellular EAs, we highlight the work of Giacobini et al. [119] as the most interesting one because, besides being the most accurate one, it considers the possibility of using both different shapes in the population and different selection operators. However, we have extended the study presented in [119] to proportional selection mechanisms (roulette wheel and linear ranking), as it is suggested in the future research lines of [119]. In our study we found that the model was not very accurate for proportional selection mechanisms, thus obtaining high error rates in most of the considered cGAs. For solving these problems we present two new mathematical models trying to improve the behavior of the existing ones. On the one hand, a polynomial model has been proposed which is based on the ratio between the shapes of the population and the neighborhood. This model is extremely accurate, but neither it is intuitive enough nor it considers the chance of using other selection schemes different from binary tournament, for which it was designed. On the other hand, a new probabilistic model has been proposed which adjusts cGAs with a much higher accuracy than the model of Giacobini et al., as shown in the studies of Chapter 5, wherein this was validated over 75 different cGAs. Additionally, this newly proposed model is capable of using both any population size and shape, and any existing selection method.

2. **Design of new cGAS with adaptive population.** As it has been commented many times through this manuscript, due to the way in which population is structured in cGAs, the speed of convergence of the algorithm is much slower than in other GAs (both structured and non-structured). The pursued effect is to keep the diversity of solutions in the population for longer but, at the same time, this also means performing a higher number of function evaluations to solve a given optimization problem. In order to address this drawback of cGAs, a new adaptive technique has been proposed which allows a cGA to enhance either the exploration or the exploitation of the most promising regions of the search space based on the convergence speed of the population. This technique lies in modifying the shape of the population, what highly affects the behavior of the algorithm (as shown in Chapter 5). Moreover, this technique has also the advantage of requiring a very low computational cost, contrary to other adaptive methods in the literature, which involve

such a highly expensive computational tasks that it is not worth applying them. All the studied cGAs with adaptive populations improve the behavior of the compared equivalent cGAs (with different static and dynamic pre-programmed populations).

3. **Design of cGAs with hierarchical population.** In the same line as the previous item, this contribution is aimed at reaching a new balance between the exploration and the exploitation of the search space for a classical cGA with the goal of improving its behavior. In this case, we propose to set up a hierarchy in the population of individuals so that the best individuals are located in the same zone of the population, thus promoting the exploitation of the most promising regions of the search space. On the other hand, the rest of individuals is also placed at different zones of the population according to their fitness values, so that exploration is promoted in these zones of the population because lower quality individuals very different from each other are put together, what allows the diversity to be kept for longer. In the study performed in this thesis, it has been proved that the cGAs with hierarchical population outperforms a canonical cGA for a wide set of problems with different features.

4. **Design of a new cellular estimation of distribution algorithm.** Our contribution here is focussed on taking advantage of the features of cGAs in the field of EDAs. Indeed, this work presents a cellular EDA (cEDA) for the first time in a PhD thesis, and we provide a preliminary study which proves that all the newly proposed cEDAs outperform their panmitic counterpart. For that, we have analyzed the behavior of the algorithms over a set of diverse problems with different features. Besides, a theoretical analysis has been also carried out by means of the study of the selection pressure, in which it has been shown that cEDAs have a lower selection pressure than the corresponding EDA, thus keeping the diversity in the population for longer, and therefore avoiding converging prematurely to local optima.

5. **Design of a new cellular memetic algorithm.** In this thesis, the design of a cellular algorithm within the important field of memetic algorithms [211] is presented for the first time in the literature. This way, we have developed specialized highly cellular algorithms by using deep problem domain knowledge. This specialization can be seen as a way for controlling the relationship between the exploration and exploitation of the algorithm in the search space. We have reached very impressive results by using this kind of algorithms. Indeed, some of these results have became the state-of-the-art for both the satisfiability problem (SAT) and the vehicle routing problem (VRP).

6. **Design of new parallel models of cGAs.** Two new models of parallel cGAs have been developed in this work. The first one has been designed for working in local area networks (LAN), and it has been compared against several parallel GA models (with both centralized and distributed in islands populations) on the SAT problem, outperforming all of them. Additionally, we have presented an island model in which each island executes a cGA. The model is intended to work on *computational grids*. With this algorithm, the state-of-the-art in many instances of the VLSVRP (*very large scale VRP*) benchmark have been improved. This benchmark is composed of the largest existing instances for that problem.

7. **Design of a new multi-objective cGA.** Another important contribution of this work has been the adaptation of the cellular model to the field of multi-objective optimization. To the best of out knowledge, no canonical multi-objective cGA existed until the publication of our study. There were only two published works in which some features of the cGA model were used [164, 178]. We have compared the new multi-objective cGA against NSGA-II and SPEA2, the state-of-the-art in

multi-objective algorithms. In this study, we have used a wide benchmark taken from the specialized literature and the results show that our algorithm outperforms both NSGA-II and SPEA2 in terms of the diversity of the set of non-dominated solutions found, while keeping a similar performance in terms of convergence with respect to the optimal Pareto front.

8. **Applications of cGAs.** Our models have been validated over a wide set of problems coming from both the combinatorial and the numerical optimization fields. Moreover, we have also considered problems involving variables of the two domains (discrete and continuous) at the same time, as well as problems with constraints. Using such a wide set of problems, some of them being real-world problems, along with the statistical analysis of the results, have allowed us to sustain our conclusions rigorously in contrast to other existing works.

9. **Design of JCell.** Finally, we have developed a library of cellular GAs, called JCell, which has been written in Java. By using JCell, any interested researcher will be easily capable of reproducing most of the experiments performed here. In addition, JCell provides the research community with the ability of further studying in depth any of the investigations developed in this work, and also to export any of the presented models to other types of algorithms or even to different research fields. With all this in mind, we consider that the implementation of JCell, together with its public availability, is an important contribution to the research community.

In summary, in this thesis we have carried out many contributions to the field of the cellular evolutionary algorithms either in the theoretical, algorithmic, or application level. From a theoretical point of view, the best existing mathematical model for approximating the selection pressure curve of a cGA has been presented. Regarding the new algorithmic models, we propose many newly different cGAs, like self-adaptive ones, those using a hierarchical structured population, and several parallel models. Besides, we have also presented new algorithmic families such as cellular EDAs, cellular memetic algorithms, and multi-objective cGAs. Both the new algorithmic models and the families of algorithms introduced here for the first time were compared against their panmitic counterparts and/or the algorithms of the state-of-the-art. In this comparison we used a large set of problems with different features and the drawn conclusions are supported with statistical confidence.

Finally, we performed wide studies of some of the newly proposed algorithms on problems coming directly from the real world, belonging to very diverse areas such as logistics or telecommunications. This way, we expose the usefulness of our models to face optimization problems which emerge both from the academy and the industry.

Trabajo Futuro

Muchas son las líneas de trabajo que surgen de las investigaciones realizadas en esta tesis doctoral. A continuación se destacan las que consideramos que gozan de un mayor interés para la comunidad científica.

En relación a los estudios teóricos realizados sobre cGAs, creemos que son interesantes líneas de trabajo futuro investigar en nuevos modelos para aproximar otros tipos de EAs, como los HcGAs o los cEDAs, estudiados en los capítulos 7 y 8, respectivamente. Otro aspecto importante será investigar en modelos más simples que el probabilístico presentado en el Capítulo 5, y que sean capaces de ajustarse con una precisión igual o superior a la curva de la presión de selección de un cGA.

En cuanto al nuevo modelo adaptativo (Capítulo 6), proponemos la inclusión de nuevos criterios más sofisticados para la adaptación, como el uso de otras técnicas basadas en el valor de fitness, u otros indicadores que pudieran describir significativamente la evolución de la población. En cualquier caso, es deseable que se utilicen criterios simples de medir, para no convertir la ventaja de la adaptación en una desventaja debido a la sobrecarga introducida en el tiempo real de ejecución. También puede resultar interesante aplicar este modelo adaptativo a otras aportaciones de este trabajo, como los cEDAs, cMAs, o MOCell. Finalmente, consideramos que es interesante para este modelo el estudio de la influencia de otros mecanismos de reubicación de individuos en la población cuando se produce un cambio en la forma de la rejilla.

También se nos presentan varias líneas de trabajo futuro interesantes relacionadas con los HcGAs propuestos en el Capítulo 7. En concreto, nuestro esfuerzo se centrará en el análisis de otras formas de jerarquías, con respecto a la forma o al criterio que permite ir ascendiendo en niveles. Además, podríamos considerar el uso de diferentes parametrizaciones del algoritmo dependiendo de la posición del individuo en cuestión, para enfatizar de esta manera distintas estrategias de búsqueda en diferentes niveles de jerarquía. También estamos llevando a cabo estudios del comportamiento de los HcGAs con diferentes políticas de actualización asíncronas.

En el Capítulo 8 hemos hecho una modesta incursión en el desarrollo de un nuevo y amplio campo de búsqueda (el de los EDAs celulares), y por tanto existe un gran número de posibles líneas de investigación futuras. En concreto, sugerimos algunos trabajos directos a raíz del estudio aquí presentado, como el análisis de otras alternativas propuestas (pero no estudiadas) en [193] sobre los esquemas de aprendizaje para los cEDAs, el ajuste fino de los parámetros utilizados, la comparación de nuestros cEDAs con otros algoritmos existentes en la literatura, y el estudio del comportamiento de otros modelos celulares basados en EDAs distintos de MUDA. Existen además otras líneas de trabajo futuro como el estudio de los efectos de la sincronización en la actualización de la población, la adaptación del algoritmo a un modelo paralelo o la adoptación de mejoras algorítmicas existentes en otros EAs celulares, como algunas de las propuestas en esta tesis.

En relación a los cMAs (Capítulo 9), sería interesante realizar un profundo estudio de ajuste de la parametrización para tratar de mejorar los resultados obtenidos. Además, la hibridación de algunos modelos mejorados de cGAs (asíncronos [20], adaptativos [15], etcétera) nos puede conducir a obtener aún mejores resultados. También, un trabajo interesante sería resolver instancias de mayor tamaño con estos algoritmos, puesto que el cMA reportó los mejores resultados en los problemas de mayor tamaño. Finalmente, estamos considerando la aplicación de cMAs sobre otros problemas complejos. De hecho, se está trabajando actualmente en este sentido, y como resultado existe ya una primera publicación del uso de cMAs para resolver el problema de la planificación de tareas en un sistema de computación *grid* [298]. En este trabajo, el cMA obtiene planificaciones de muy alta calidad en relación con los algoritmos considerados, tomados de otras publicaciones en el campo.

Sería también interesante comparar los modelos paralelos propuestos en el Capítulo 10 con algunos de los principales modelos existentes en función de la escalabilidad, de la carga en las comunicaciones, y de los resultados y tiempos obtenidos. Además, en relación al estudio de PEGA sobre las instancias VLSVRP, estamos trabajando en incrementar el número de ejecuciones de nuestras pruebas, ya que el resultado de tan pocas ejecuciones no puede considerarse relevante en algoritmos de naturaleza estocástica como el nuestro. También se está poniendo esfuerzo en la posibilidad de reducir los tiempos de ejecución. Esto se puede conseguir incrementando el número de ordenadores de nuestro sistema de computación *grid*, o modificando el algoritmo para hacerlo más eficiente. En relación a este segundo aspecto, creemos que debemos centrarnos en desarrollar nuevos métodos de búsqueda local más eficientes que el utilizado en este trabajo, ya que es el paso más costoso computacionalmente del ciclo reproductor del algoritmo.

En cuanto al estudio de cGAs sobre problemas multi-objetivo (Capítulo 11), la evaluación de MOCell con otros conjuntos de problemas (con dos o más de dos objetivos) y su aplicación para resolver los problemas del mundo real son un asunto de trabajo futuro. Además, existe también la posibilidad de investigar nuevos aspectos de diseño de MOCell que puedan mejorar su comportamiento, como por ejemplo el uso de otras políticas asíncronas de actualización de individuos.

También existen algunas líneas abiertas de trabajo futuro al estudio realizado en el Capítulo 13 sobre el problema VRP. En concreto, estamos interesados en reducir los tiempos de ejecución del algoritmo; para ello probaremos el uso de otros métodos de búsqueda local (como por ejemplo 3-Opt), o aplicar esta etapa de una manera menos exhaustiva. Otro paso futuro sería resolver otras variantes del problema más cercanas al mundo real, como VRP con ventanas de tiempo (VRPTW), con múltiples almacenes (MDVRP), o con recogida de artículos sobrantes (VRPB). En esta línea, se han realizado ya algunas investigaciones [12], en las que se ha definido un nuevo tipo de problema VRP dinámico que trata de simular algunos de los cambios a los que se enfrentan frecuentemente las empresas logísticas del mundo real, como por ejemplo modificaciones en la localización del almacén o en las flotas de vehículos utilizadas.

En cuanto al estudio de los cGAs en el dominio numérico (Capítulo 14), proponemos como trabajo futuro incluir las diferentes técnicas propuestas en el Capítulo 6 [15] para mejorar la relación de exploración/explotación del algoritmo, lo que nos podría llevar a mejorar los resultados actuales. Además, sería interesante extender el estudio sobre el comportamiento de JCell en optimización continua probando, por ejemplo, otros operadores de recombinación y mutación especializados.

Finalmente, también surgen líneas de trabajo futuro relacionadas con el estudio de los problemas DFCNT y cDFCNT del Capítulo 15. En particular, se está llevando a cabo la aplicación de MOCell, la extensión de cMOGA propuesta en el Capítulo 11, para estos dos problemas, con el objetivo de mejorar los resultados del Capítulo 15. También estamos planeando realizar un análisis más profundo en el modelo de búsqueda de MOCell, y pretendemos paralelizar el algoritmo para reducir los tiempos de ejecución a valores prácticos. De esta manera, podremos aumentar el área de simulación de las redes estudiadas. Finalmente, otro inmediato e interesante paso futuro consistiría en el diseño y estudio de nuevos entornos que simulen otros escenarios interesantes del mundo real como el diseño óptimo de redes vehiculares.

Por último, se pretende realizar una publicación importante que permita dar a conocer a la comunidad científica internacional el trabajo desarrollado aquí, entre otras cosas. Para ello, se ha firmado un contrato con Springer para la publicación de un volumen que se titulará *Cellular Genetic Algorithms*. Está previsto que este nuevo libro esté disponible a finales de 2007.

Parte V

Apéndices

Apéndice A

Otros Problemas Estudiados

Con el fin de hacer este documento de tesis autocontenido, presentamos en este capítulo una breve descripción de todos los problemas que hemos utilizado en nuestros estudios. Para ello, hemos estructurado en distintas secciones la descripción de todos los problemas dependiendo del campo de optimización al que pertenezcan: optimización combinatoria en la Sección A.1, optimización continua en la Sección A.2, y optimización multi-objetivo en la Sección A.3.

A.1. Problemas de Optimización Combinatoria

En esta sección describimos los distintos problemas de optimización combinatoria que hemos abordado durante nuestro trabajo de tesis. En la definición de algunos de ellos se utiliza la distancia de Hamming. La distancia de Hamming entre una cadena de bits a y otra b se mide con la ecuación:

$$d_{ab} = \sum_{i=1}^{l} a_i \otimes b_i \ . \tag{A.1}$$

Por tanto, cadenas de bits idénticas tienen una distancia de Hamming de $d_{ab} = 0.0$, mientras que cadenas de bits completamente diferentes tendrán una distancia de $d_{ab} = l$, siendo l la longitud de las dos cadenas.

A.1.1. Problema COUNTSAT

El problema COUNTSAT [89] es una instancia de MAXSAT. En COUNTSAT el valor de la solución se corresponde con el número de cláusulas (de entre todas las posibles cláusulas de Horn de tres variables) satisfechas por una entrada formada por n variables booleanas. Es fácil comprobar que el óptimo se consigue cuando todas las variables tienen el valor 1. En este estudio consideramos una instancia de $n = 20$ variables, y el valor de la solución óptima al problema es 6860 (Ecuación A.2).

$$\begin{aligned} f_{\text{COUNTSAT}}(s) &= s + n(n-1)(n-2) - 2(n-2)\left(\begin{array}{c} s \\ 2 \end{array}\right) + 6\left(\begin{array}{c} s \\ 3 \end{array}\right) \ , \\ &= s + 6840 - 18s(s-1) + s(s-1)(s-2) \ . \end{aligned} \tag{A.2}$$

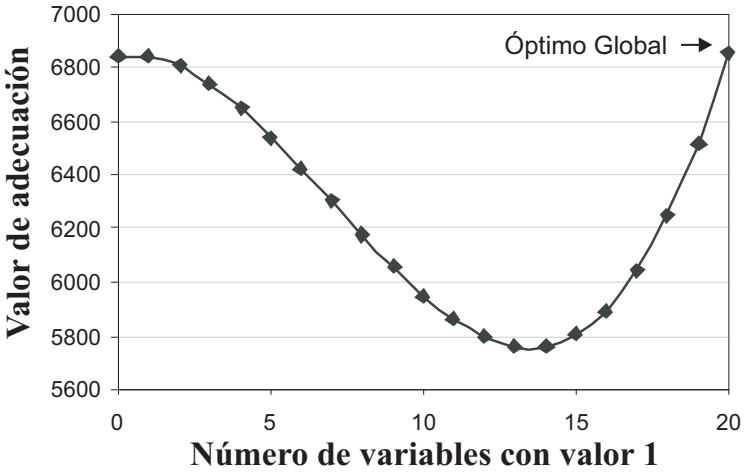


Figura A.1: Función COUNTSAT con 20 variables.

La función COUNTSAT está extraída de MAXSAT con la intención de ser muy difícil de resolver con GAs [89]. Las entradas aleatorias generadas uniformemente tendrán $n/2$ unos en término medio. Por tanto, los cambios locales decrementando el número de unos nos conducirán hacia entradas mejores, mientras que si se incrementa el número de unos se decrementará el valor de adecuación. En consecuencia, cabe esperar que un GA encuentre rápidamente la cadena formada por todos sus componentes a cero y tenga dificultades para encontrar la cadena compuesta por unos en todas sus posiciones.

A.1.2. Problema del Diseño de Códigos Correctores de Errores – ECC

El problema ECC se presentó en [192]. Consideremos una tupla (n, M, d) donde n es la longitud (número de bits) de cada palabra de código, M es el número de palabras de código, y d es la mínima distancia de Hamming entre dos palabras de código cualesquiera. El objetivo será obtener un código con el mayor valor posible de d (reflejando una mayor tolerancia al ruido y a los errores), y con valores de n y M fijados previamente. La función de adecuación que maximizaremos se corresponde con:

$$f_{\text{ECC}}(C) = \frac{1}{\sum_{i=1}^{M} \sum_{\substack{j=1 \\ i \neq j}}^{M} \frac{1}{d_{ij}^2}} \quad , \tag{A.3}$$

donde d_{ij} representa la distancia de Hamming entre las palabras i y j del código (compuesto de M palabras de longitud n). En este estudio consideramos una instancia donde C está formado por $M = 24$ palabras de longitud $n = 12$ bits. Lo que forma un espacio de búsqueda de tamaño $\binom{4096}{24}$, que es 10^{87} aproximadamente. La solución óptima para la instancia con $M = 24$ y $n = 12$ tendrá el valor de adecuación 0.0674 [56]. En algunos de nuestros estudios, hemos simplificado el espacio de búsqueda del problema reduciendo a la mitad $(M/2)$ el número de palabras que forman el código, y la otra mitad se compone del complemento de las palabras encontradas por el algoritmo.

A.1.3. Problema de la Modulación en Frecuencia de Sonidos – FMS

El problema FMS [278] consiste en determinar los 6 parámetros reales $\vec{x} = (a_1, w_1, a_2, w_2, a_3, w_3)$ del modelo de sonido en frecuencia modulada dado en la Ecuación A.4 para aproximarla a la onda de sonido dada en la Ecuación A.5, donde $\theta = 2\pi/100$. Este problema puede ser definido como un problema de optimización discreta o continua. En el caso que discreto, los parámetros están codificados en el rango $[-6.4, +6.35]$ con cadenas de 32 bits. El caso continuo está descrito en la Sección A.2.

$$y(t) = a_1 \sin\Big(w_1 t\theta + a_2 \sin\big(w_2 t\theta + a_3 \sin(w_3 t\theta)\big)\Big) \ , \tag{A.4}$$

$$y_0(t) = 1.0 \sin\Big(5.0 t\theta - 1.5 \sin\big(4.8 t\theta + 2.0 \sin(4.9 t\theta)\big)\Big) \ . \tag{A.5}$$

El objetivo de este problema consiste en minimizar la suma del error cuadrático medio entre los datos de muestreo (Ecuación A.4) y los reales (Ecuación A.5), como se detalla en la Ecuación A.6. Este problema es una función multimodal altamente compleja, y fuertemente epistática también. Debido a la extrema dificultad para resolver este problema con alta precisión sin utilizar operadores específicos de optimización continua (como el GA gradual [142]), el algoritmo se detiene cuando el error cae por debajo de 10^{-2}. La función de adecuación que tendremos que maximizar se corresponde con la inversa de la Ecuación A.6, y obtenemos su máximo valor cuando $E_{FMS} = 0.0$.

$$E_{\text{FMS}}(\vec{x}) = \sum_{t=0}^{100} \Big(y(t) - y_0(t)\Big)^2 \ . \tag{A.6}$$

A.1.4. Problema IsoPeak

IsoPeak es un problema no separable que fue investigado en [194]. Las soluciones para esta función están compuestas por un vector n-dimensional, de forma que $n = 2 \times m$ (los genes están divididos en grupos de dos). En primer lugar, definimos dos funciones auxiliares $Iso1$ e $Iso2$ como:

$\overrightarrow{x}$	**00**	**01**	**10**	**11**
$Iso1$	m	0	0	$m-1$
$Iso2$	0	0	0	m

Ahora, podemos definir el problema IsoPeak como:

$$f_{\text{IsoPeak}(\overrightarrow{x})} = Iso2\,(x_1, x_2) + \sum_{i=2}^{m} Iso1\,(x_{2i-1}, x_{2i}) \ . \tag{A.7}$$

El objetivo del problema es maximizar la función f_{IsoPeak} y el óptimo global se alcanza en el caso de que las variables del vector n-dimensional tomen los valores $(1, 1, 0, 0, \ldots, 0, 0)$.

A.1.5. Problema del Máximo Corte de un Grafo – MAXCUT

El problema MAXCUT consiste en dividir un grafo ponderado $G = (V, E)$ en dos sub-grafos disjuntos $G_0 = (V_0, E_0)$ y $G_1 = (V_1, E_1)$ de manera que la suma de los pesos de los ejes que posean un extremo en V_0 y el otro en V_1 sea máxima. Para codificar una partición de los vértices utilizaremos una cadena binaria $(x_1, x_2, \ldots, x_n)$ donde cada dígito se corresponde con un vértice. De manera que si un dígito

tiene valor 1, su vértice correspondiente estará en el conjunto V_1, y si tuviera valor 0 el vértice respectivo pertenecería a V_0. La función que maximizaremos es [161]:

$$f_{\text{MAXCUT}}(\vec{x}) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} w_{ij} \cdot \left[x_i(1-x_j) + x_j(1-x_i) \right] \ . \tag{A.8}$$

Téngase en cuenta que w_{ij} contribuye a la suma sólo en el caso de que los nodos i y j se encuentren en distintas particiones. Aunque es posible generar instancias de grafos diferentes de forma aleatoria, hemos utilizado tres instancias distintas del problema tomadas de [161]. Dos de ellas se corresponden con grafos generados aleatoriamente: uno disperso "cut20.01" y otro denso "cut20.09"; ambos formados por 20 vértices. La otra instancia es un grafo escalable de 100 vértices. Las soluciones óptimas para estas instancias son 10.119812 para el caso de "cut20.01", 56.740064 para "cut20.09" y 1077 para "cut100".

A.1.6. Problema Masivamente Multimodal – MMDP

El problema MMDP ha sido específicamente diseñado para ser difícil de afrontar para los EAs [123]. Está compuesto de k sub-problemas engañosos (s_i) de 6 bits cada uno. El valor de cada uno de estos sub-problemas (fitness_{s_i}) depende del número de unos que tiene la cadena binaria que lo compone (ver la Figura A.2). Es fácil comprender por qué se considera a esta función engañosa, puesto que estas sub-funciones tienen dos máximos globales y un atractor engañoso en el punto medio (gráfica de la Figura A.2).

Nº de Unos	Valor de la Subfunción
0	1.000000
1	0.000000
2	0.360384
3	0.640576
4	0.360384
5	0.000000
6	1.000000

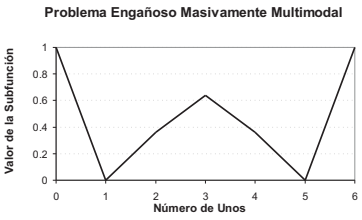


Figura A.2: Función bipolar engañosa básica (s_i) para MMDP.

En MMDP, cada sub-problema s_i contribuye al valor de fitness total en función del número de unos que tiene (Figura A.2). El óptimo global del problema tiene el valor k, y se alcanza cuando todos los sub-problemas están compuestos de cero o seis unos. El número de óptimos locales es muy grande (22^k), mientras que sólo hay 2^k soluciones globales. Por tanto, el grado de multimodalidad viene dado por el parámetro k. En esta tesis, utilizaremos (mientras no se especifique lo contrario) una instancia considerablemente grande de $k = 40$ sub-problemas. La función que trataremos de maximizar para resolver el problema se muestra en la Ecuación A.9, y su valor máximo es 40.

$$f_{\text{MMDP}}(\vec{s}) = \sum_{i=1}^{k} \text{fitness}_{s_i} \ . \tag{A.9}$$

A.1.7. Problema de las Tareas de Mínima Espera – MTTP

El problema MTTP [262] es un problema de planificación de tareas en el que cada tarea i del conjunto de tareas $T = \{1, 2, \ldots, n\}$ tiene una longitud l_i –el tiempo que tarda en ejecutarse– un límite de tiempo (o *deadline*) d_i –antes del cual debe ser planificada– y un peso w_i. El peso es una penalización que debe ser añadida a la función objetivo en el caso de que la tarea permanezca sin estar planificada. Las longitudes, pesos, y límites de tiempo de las tareas son todos números enteros positivos. Planificar la tarea de un sub-conjunto S de T consiste en encontrar el tiempo de comienzo de cada tarea en S, de forma que como mucho se realiza una tarea cada vez y que cada tarea termina de realizarse antes de su límite de tiempo.

Definimos una función de planificación g sobre $S \subseteq T : S \mapsto \mathbb{Z}^+ \bigcup \{0\} \mid \forall\ i,\ j \in S$ con estas dos propiedades:

1. Una tarea no puede ser planificada antes de que haya terminado alguna otra ejecutada previamente: $g(i) < g(j) \Rightarrow g(i) + \ l_i \leq g(j)$.

2. Toda tarea tiene que finalizar antes de su límite de tiempo: $g(i) + l_i \leq d_i$.

El objetivo del problema es minimizar la suma de los pesos de las tareas no planificadas (Ecuación A.10). Por tanto, la planificación óptima minimiza la función:

$$W = \sum_{i \in T - S} w_i \ . \tag{A.10}$$

La planificación S está representada mediante un vector $\vec{x} = (x_1, x_2, \ldots, x_n)$ que contiene todas las tareas ordenadas según su límite de tiempo para ser realizada. Cada $x_i \in \{0, 1\}$, donde si $x_i = 1$ entonces la tarea i está contenida en la planificación S, mientras que si $x_i = 0$ la tarea i no está incluida en S. La función que optimizaremos, descrita en [161], es la inversa de la Ecuación A.10: $f_{\text{MTTP}} = 1/W$. Hemos utilizado para nuestros estudios tres diferentes instancias [161]: "mttp20", "mttp100" y "mttp200", de tamaños 20, 100 y 200, y valores óptimos de 0.02439, 0.005 y 0.0025, respectivamente.

A.1.8. Problema OneMax

El problema OneMax [248] (o recuento de bits) es un problema muy simple que consiste en maximizar el número de unos que contiene una cadena de bits. Formalmente, este problema se puede describir como encontrar una cadena $\vec{x} = \{x_1, x_2, \ldots, x_n\}$ con $x_i \in \{0, 1\}$, que maximice la siguiente ecuación:

$$f_{\text{OneMax}}(\vec{x}) = \sum_{i=1}^{n} x_i \ . \tag{A.11}$$

La función f_{OneMax} tiene $(n+1)$ posibles valores diferentes, que están distribuidos multimodalmente. Para las *funciones aditivamente descomponibles* (ADFs) la distribución multimodal ocurre con bastante frecuencia [124]. Para definir una instancia de este problema, sólo necesitamos definir la longitud de la cadena de bits (n). Para un n dado, la solución óptima al problema es una cadena de n unos, es decir, se les fija el valor uno a todos los bits de la cadena.

A.1.9. Problema Plateau

Este problema fue estudiado en [205], y es también conocido como el problema del camino real de 3 bits. Las soluciones para esta función consisten en un vector n-dimensional, de forma que $n = 3 \times m$ (los genes se dividen en grupos de tres). Primero, definimos una función auxiliar g como:

$$g(x_1, x_2, x_3) = \begin{cases} 1, & \text{if} \quad x_1 = 1 \quad \text{and} \quad x_2 = 1 \quad \text{and} \quad x_3 = 1 \\ 0, & \text{otherwise} \ . \end{cases} \tag{A.12}$$

Ahora, podemos definir el problema Plateau como:

$$f_{\text{Plateau}}(\overrightarrow{x}) = \sum_{i=1}^{m} g(\overrightarrow{s_i}) \quad . \tag{A.13}$$

Donde $\overrightarrow{s_i} = (x_{3i-2}, x_{3i-1}, x_{3i})$. El objetivo del problema es maximizar la función f_{Plateau}, y el óptimo global se obtiene cuando todos los bits de la cadena están a uno.

A.1.10. Problema P-PEAKS

El problema P-PEAKS [157] es un generador de problemas multimodales. Un generador de problemas es una tarea fácilmente parametrizable, con un grado de dificultad que se puede afinar, de manera que nos permita generar instancias con una complejidad tan grande como queramos. Adicionalmente, el uso de un generador de problemas elimina la posibilidad de afinar a mano los algoritmos para un problema particular, permitiendo de esta forma una mayor justicia al comparar los algoritmos. Utilizando un generador de problemas, evaluamos nuestros algoritmos sobre un elevado número de instancias de problema aleatorias, ya que cada vez que se ejecuta el algoritmo resolvemos una instancia distinta. Por tanto, se incrementa el poder de predicción de los resultados para la clase de problemas como tal, no para instancias concretas.

La idea de P-PEAKS consiste en generar P cadenas aleatorias de N bits que representan la localización de P cimas en el espacio de búsqueda. El valor de adecuación de una cadena se corresponde con la distancia de Hamming de esa cadena con respecto a la cima más próxima, dividida por N (como se muestra en la Ecuación A.14). Usando un mayor (o menor) número de cimas obtenemos problemas más (o menos) epistáticos. En nuestro caso hemos utilizado una instancia de $P = 100$ cimas de longitud $N = 100$ bits cada una, representando un nivel de epistasis medio/alto [28]. El máximo valor de fitness que podremos conseguir es 1.0.

$$f_{\text{P-PEAKS}}(\vec{x}) = \frac{1}{N} \max_{i=1}^{P} \{N - \text{Hamming}(\vec{x}, \text{Peak}_i)\} \quad . \tag{A.14}$$

A.1.11. Problema de Satisfacibilidad – SAT

El problema de satisfacibilidad (SAT) ha recibido una gran atención por parte de la comunidad científica ya que juega un papel principal en la NP-completitud de los problemas [111]. Esto es debido a que se ha demostrado que cualquier problema puede traducirse a un problema SAT equivalente en tiempo polinómico (teorema de Cook) [62]; mientras que la transformación inversa no siempre existe en tiempo polinomial. El problema SAT fue el primero que se pudo demostrar que pertenece a la clase de problemas NP.

El problema SAT consiste en asignar valores a un conjunto de n variables booleanas $x = (x_1, x_2, \ldots, x_n)$ de forma que satisfagan un conjunto dado de cláusulas $c_1(\vec{x}), \ldots, c_m(\vec{x})$, donde $c_i(\vec{x})$ es una disyunción de literales, y un literal es una variable o su negación. Por tanto, podemos definir SAT como una función $f : \mathbb{B}^n \rightarrow \mathbb{B}$, siendo $\mathbb{B} = \{0, 1\}$, tal que:

$$f_{\text{SAT}}(\vec{x}) = c_1(\vec{x}) \wedge c_2(\vec{x}) \wedge \ldots \wedge c_m(\vec{x}) \ . \tag{A.15}$$

Una solución al problema SAT, $\vec{x}$, se dice que es satisfacible si $f_{SAT}(\vec{x}) = 1$, e insatisfacible en otro caso. Una instancia del problema k-SAT está formada únicamente por cláusulas de longitud k. Cuando $k \geq 3$ el problema es NP-completo [111]. En esta tesis consideramos únicamente instancias de 3-SAT que pertenezcan a la fase de transición de dificultad del problema SAT, a la que pertenecen las instancias más duras, y que verifican que $m = 4.3 * n$ [209] (siendo m el número de cláusulas y n el de variables). En concreto, durante los estudios de esta tesis se han utilizado los conjuntos de 12 instancias (con $n = 30$ hasta 100 variables) propuestos en [39], pertenecientes a la fase de transición de dificultad del problema SAT.

Según la Ecuación A.15, una función de fitness sencilla para SAT consiste en contar el número de cláusulas que satisface la solución a evaluar. Esta función tiene el problema de que asigna el mismo valor de fitness a múltiples soluciones distintas. Una alternativa consiste en considerar la función de fitness como una función lineal del número de cláusulas satisfechas en la que utilizamos la *adaptación de pesos paso a paso* (*stepwise adaptation of weights* (SAW)) [95]:

$$f_{\text{SAW}}(\vec{x}) = w_1 \cdot c_1(\vec{x}) + \ldots + w_m \cdot c_m(\vec{x}) \ . \tag{A.16}$$

Esta función pondera los valores de las cláusulas con $w_i \in \mathbb{B}$ para darle más importancia a aquellas cláusulas que no han sido satisfechas aún por la mejor solución actual. Estos pesos son ajustados dinámicamente de acuerdo con la ecuación $w_i = w_i + 1 - c_i(\vec{x}^*)$, siendo $\vec{x}^*$ el mejor individuo actual.

A.2. Problemas de Optimización Continua

Se presentan en esta sección los problemas pertenecientes al campo de la optimización continua que hemos estudiado durante este trabajo de tesis. Algunos de estos problemas son funciones académicas clásicas bien conocidas (sección A.2.1), mientras que otras están tomadas del mundo real (sección A.2.2).

A.2.1. Problemas Académicos

Como ya se ha dicho anteriormente, hemos utilizado durante los estudios de esta tesis ocho problemas académicos de optimización continua. Éstos son las funciones de *Griewangk* (f_{Gri}) [139], *Rastrigin generalizada* (f_{Ras}) [36, 274], *Rosenbrock generalizada* (f_{Ros}) [75], *Schwefel 1.2* (f_{Sch}) [252], el modelo *Sphere* (f_{Sph}) [75, 252], la *expansión de* f_{10} (ef_{10}) [289], y las funciones *Fractal* (f_{Frac}) [37] y de *Ackley* (f_{Ack}), originalmente propuesta por Ackley [5] como una función bidimensional, y generalizada posteriormente por Bäck et al. [41]. Este conjunto de problemas incorpora muy diversas características, ya que está compuesto por problemas lineales y no lineales, unimodales y multimodales, escalables y no escalables, convexos, etc. Todos los detalles de estos problemas se muestran en la Tabla A.1, donde n es el tamaño del problema que hemos utilizado normalmente en este trabajo. El valor de la solución óptima para todos los problemas de la Tabla A.1 es 0.0.

Durante nuestros tests, nos hemos dado cuenta de que existe una importante pérdida de precisión cuando se calculan los valores de fitness de las funciones Griewangk y Rastrigin cerca del valor óptimo. La

Tabla A.1: Conjunto de problemas académicos de optimización continua.

Problema	**Función de fitness**	**n**	**Valores de las vbles.**
Griewangk	$f_{\text{Gri}}(\vec{x}) = \frac{1}{d}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$ $d = 4000$	25	$-600.0 \leq x_i \leq 600.0$
Rastrigin	$f_{\text{Ras}}(\vec{x}) = a \cdot n + \sum_{i=1}^{n} x_i^2 - a \cdot \cos(\omega \cdot x_i)$ $a = 10,\ \omega = 2\pi$	25	$-5.12 \leq x_i \leq 5.12$
Rosenbrock	$f_{\text{Ros}}(\vec{x}) = \sum_{i=1}^{n-1}(100 \cdot (x_{i+1} - x_i^2)^2 + (x_i - 1)^2)$	25	$-5.12 \leq x_i \leq 5.12$
Schwefel	$f_{\text{Sch}}(\vec{x}) = \sum_{i=1}^{n}\left(\sum_{j=1}^{i} x_j\right)^2$	25	$-65.536 \leq x_i \leq 65.536$
Sphere	$f_{\text{Sph}}(\vec{x}) = \sum_{i=1}^{n} x_i^2$	25	$-5.12 \leq x_i \leq 5.12$
ef_{10}	$f_{\text{ef}_{10}}(\vec{x}) = f_{10}(x_1, x_2) + \ldots + f_{10}(x_{i-1}, x_i) + \ldots + f_{10}(x_n, x_1)$ $f_{10}(x, y) = (x^2 + y^2)^{0.25} \cdot \left[\sin^2(50 \cdot (x^2 + y^2)^{0.1}) + 1\right]$	10	$-100.0 < x_i \leq 100.0$
Fractal	$f_{\text{Frac}}(\vec{x}) = \sum_{i=1}^{n}(C'(x_i) + x_i^2 - 1)$ $C'(z) = \begin{cases} \frac{C(z)}{C(1)\lvert z\rvert^{2-D}} & \text{si } z \neq 0 \\ 1 & \text{si } z = 0 \end{cases}$ $C(z) = \sum_{j=-\infty}^{\infty} \frac{1-\cos(b^j z)}{b^{(2-D)j}}$ $D = 1.85,\ b = 1.5$	20	$-5.00 \leq x_i \leq 5.00$
Ackley	$f_{\text{Ack}}(\vec{x}) = -a\exp\left[-b\left(\frac{1}{n}\sum_{i=1}^{n} x_i^2\right)^{1/2}\right] - exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(cx_i)\right) + a + e$ $a = 20,\ b = 0.2,\ c = 2\pi$	25	$-32.768 \leq x_i \leq 32.768$

razón es que cuando calculamos el valor de las funciones para una solución cercana a la óptima (valores de los genes muy próximos a 0.0), algunos de los sumandos de estas funciones ($\frac{1}{d}\sum_{i=1}^{n} x_i^2$ en f_{Gri} y $\sum_{i=1}^{n} x_i^2$ en f_{Ras}) son redondeados a 0.0 al calcular el valor de fitness ya que son más de 100 órdenes de magnitud menores que los otros sumandos. Con el fin de evitar este redondeo no deseable proponemos en este capítulo dos nuevas definiciones para estas funciones que nos permiten obtener valores mucho más precisos. Estas nuevas definiciones de las funciones Griewangk y Rastrigin simplemente consisten en alterar el orden de los sumandos en la función, como se muestra en las ecuaciones A.17 y A.18. En cualquier caso, los problemas que hemos resuelto durante los trabajos realizados en esta tesis son las versiones clásicas (menos precisas) con el fin de ser justos en las comparaciones con los demás algoritmos de la literatura.

$$f_{\text{Gri\_Acc}}(\vec{x}) = \left(1 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right)\right) + \frac{1}{d}\sum_{i=1}^{n} x_i^2 \ ; \tag{A.17}$$

$$f_{\text{Ras\_Acc}}(\vec{x}) = \left(a \cdot n - \sum_{i=1}^{n} a \cdot \cos(\omega \cdot x_i)\right) + \sum_{i=1}^{n} x_i^2 \ . \tag{A.18}$$

A continuación mostramos las diferencias en los resultados utilizando las dos funciones de fitness para cada problema (la precisa o la estándar) con JCell (la parametrización utilizada es la mostrada en el Capítulo 14). La comparación de los resultados de ejecutar JCell sobre estas funciones se muestra en la Tabla A.2. Como puede verse, cuando utilizamos las funciones precisas propuestas, el algoritmo no encuentra el óptimo en ninguna de las ejecuciones, mientras que en el caso de las funciones estándar (utilizando la misma parametrización de JCell) se encuentra en más del 80 % de las ejecuciones para los dos problemas. Además, tras calcular los p-valores sobre los resultados, hemos obtenido relevancia estadística en todas las diferencias, aunque los valores medios mostrados son parecidos en las dos versiones de los problemas.

Tabla A.2: Comparación entre los resultados obtenidos con las funciones de Griewangk y Rastrigin estándares y con sus nuevas definiciones más precisas.

JCell	Mejor	Media	Test
f_{Gri}	94 %	2.381E-3 $_{\pm 4.75E-3}$	+
$f_{\text{Gri\_Acc}}$	4.13E-156	2.3E-3 $_{\pm 4.75E-3}$	
f_{Ras}	82 %	2.734E-5 $_{\pm 9.299E-6}$	+
$f_{\text{Ras\_Acc}}$	2.734E-138	2.113E-6 $_{\pm 9.299E-6}$	

A.2.2. Problemas del Mundo Real

Incluimos en nuestros estudios tres problemas del mundo real para hacer más completos nuestros estudios. Estos problemas son el de la *identificación de parámetros en el problema de modulación en frecuencia de sonidos* (f_{fms}) [277], el de la resolución de *sistemas de ecuaciones lineales* (f_{Sle}) [97] y un problema de *ajuste polinomial* (f_{Cheb}) [264].

Identificación de parámetros en el problema de modulación en frecuencia de sonidos

Este problema se define determinando los 6 parámetros reales $\vec{x} = (a_1, w_1, a_2, w_2, a_3, w_3)$ del modelo de sonido modulado en frecuencia que aparece en la Ecuación A.19 para aproximarlo a la onda de sonido dada en la Ecuación A.20 (donde $\theta = 2 \cdot \pi/100$). El rango de calores que pueden tomar los parámetros está definido en el intervalo $[-6.4, +6.35]$.

$$y(t) = a_1 \cdot \sin(\omega_1 \cdot t \cdot \theta + a_2 \cdot \sin(\omega_2 \cdot t \cdot \theta + a_3 \cdot \sin(\omega_3 \cdot t \cdot \theta))) \ , \tag{A.19}$$

$$y_0(t) = 1.0 \cdot \sin(5.0 \cdot t \cdot \theta - 1.5 \cdot \sin(4.8 \cdot t \cdot \theta + 2.0 \cdot \sin(4.9 \cdot t \cdot \theta))) \ . \tag{A.20}$$

El objetivo es minimizar la suma de los errores al cuadrado dados en la Ecuación A.21. Este problema es una función multimodal muy compleja altamente engañosa, cuyo valor óptimo es 0.0.

$$f_{\text{fms}}(\vec{x}) = \sum_{t=0}^{100} \Big(y(t) - y_0(t)\Big)^2 \ . \tag{A.21}$$

Sistemas de ecuaciones lineales

Este problema consiste en encontrar los valores del vector $\vec{x}$ tal que $A\vec{x} = \vec{b}$, con:

$$A = \begin{pmatrix} 5,4,5,2,9,5,4,2,3,1 \\ 9,7,1,1,7,2,2,6,6,9 \\ 3,1,8,6,9,7,4,2,1,6 \\ 8,3,7,3,7,5,3,9,9,5 \\ 9,5,1,6,3,4,2,3,3,9 \\ 1,2,3,1,7,6,6,3,3,3 \\ 1,5,7,8,1,4,7,8,4,8 \\ 9,3,8,6,3,4,7,1,8,1 \\ 8,2,8,5,3,8,7,2,7,5 \\ 2,1,2,2,9,8,7,4,4,1 \end{pmatrix} ; \ \vec{b} = \begin{pmatrix} 40 \\ 50 \\ 47 \\ 59 \\ 45 \\ 35 \\ 53 \\ 50 \\ 55 \\ 40 \end{pmatrix} \ . \tag{A.22}$$

La función de evaluación que minimizamos en nuestros experimentos se muestra en la Ecuación A.23. Esta función tiene la solución óptima $f_{\text{sle}}(\vec{x^*}) = 0.0$, y los valores de las diez variables del problema están dentro del intervalo $[-9.0, 11.0]$.

$$f_{\text{Sle}}(\vec{x}) = \left| \sum_{i=1}^{n} \sum_{j=1}^{n} (a_{ij} \cdot x_j) - b_i \right| \quad . \tag{A.23}$$

Problema de ajuste polinomial

Para este problema, el objetivo es encontrar los coeficientes del siguiente polinomio en z:

$$P(z) = \sum_{j=0}^{2k} c_j z^j, \quad k \in \mathbb{Z}^+ \quad , \tag{A.24}$$

tal que $\forall z^j \in [-1, +1]$, $P(z) \in [-1, +1]$, $P(+1.2) \geq T_{2k}(+1.2)$, y $P(-1.2) \geq T_{2k}(-1.2)$, donde $T_{2k}(z)$ es un polinomio f_{Cheb} de grado $2k$.

La solución al problema de ajuste polinomial consta de los coeficientes de $T_{2k}(z)$. Este polinomio oscila entre -1 y +1. Fuera de esta región, el polinomio aumenta precipitadamente en la dirección positiva de los valores del eje de ordenadas. Este problema tiene sus raíces en el diseño de un filtro electrónico y reta a un proceso de optimización forzándolo a que encuentre valores de parámetros con diversas magnitudes, algo que es muy común en los sistemas industriales. El polinomio f_{Cheb} empleado aquí es:

$$T_8(z) = 1 - 32z^2 + 160z^4 - 256z^6 + 128z^8 \quad . \tag{A.25}$$

Es un problema de nueve parámetros ($\vec{x} = [x_1, \ldots, x_9]$). Es necesario realizar una pequeña corrección para transformar las restricciones de este problema en una función objetivo a minimizar, llamada f_{Cheb} (véase [142] para más detalles). Cada parámetro (coeficiente) se encuentra en el intervalo [-5.12,+5.12]. El valor del óptimo la función objetivo es $f_{\text{Cheb}}(\vec{x^*}) = 0.0$.

A.3. Problemas de Optimización Multi-objetivo

Presentamos en esta sección los conjuntos de problemas teóricos multi-objetivo que hemos estudiado. Podemos dividirlos en problemas con restricciones y sin restricciones. Los problemas sin restricciones escogidos incluyen los de Schaffer, Fonseca, y Kursawe, al igual que los problemas ZDT1, ZDT2, ZDT3, ZDT4, y ZDT6 [300]. Su formulación se muestra en la Tabla A.3. Los problemas con restricciones son Osyczka2, Tanaka, Srinivas, y ConstrEx, y se describen en la Tabla A.4. Todos estos problemas son bien conocidos en el campo de la optimización multi-objetivo, y pueden ser encontrados en libros como [60, 78].

Otro conjunto de problemas utilizado en este estudio es el obtenido por WFG, la herramienta recientemente propuesta en [151]. Esta herramienta le permite al usuario definir conjuntos de problemas con diferentes propiedades. En este estudio utilizamos la versión de dos objetivos de nueve problemas, WFG1 a WFG9. Las propiedades de estos problemas se detallan en la Tabla A.5.

Tabla A.3: Funciones de prueba sin restricciones.

Problema	Funciones objetivo	Rango de las variables	n
Schaffer	$f_1(x) = x^2$ $f_2(x) = (x-2)^2$	$-10^5 \leq x \leq 10^5$	1
Fonseca	$f_1(\vec{x}) = 1 - e^{-\sum_{i=1}^{n}\left(x_i - \frac{1}{\sqrt{n}}\right)^2}$ $f_2(\vec{x}) = 1 - e^{-\sum_{i=1}^{n}\left(x_i + \frac{1}{\sqrt{n}}\right)^2}$	$-4 \leq x_i \leq 4$	3
Kursawe	$f_1(\vec{x}) = \sum_{i=1}^{n-1}\left(-10e^{\left(-0.2*\sqrt{x_i^2+x_{i+1}^2}\right)}\right)$ $f_2(\vec{x}) = \sum_{i=1}^{n}(\lvert x_i\rvert^a + 5\sin x_i^b); a=0.8; b=3$	$-5 \leq x_i \leq 5$	3
ZDT1	$f_1(\vec{x}) = x_1$ $f_2(\vec{x}) = g(\vec{x})[1-\sqrt{x_1/g(\vec{x})}]$ $g(\vec{x}) = 1 + 9(\sum_{i=2}^{n} x_i)/(n-1)$	$0 \leq x_i \leq 1$	30
ZDT2	$f_1(\vec{x}) = x_1$ $f_2(\vec{x}) = g(\vec{x})[1-(x_1/g(\vec{x}))^2]$ $g(\vec{x}) = 1 + 9(\sum_{i=2}^{n} x_i)/(n-1)$	$0 \leq x_i \leq 1$	30
ZDT3	$f_1(\vec{x}) = x_1$ $f_2(\vec{x}) = g(\vec{x})\left[1-\sqrt{\frac{x_1}{g(\vec{x})}} - \frac{x_1}{g(\vec{x})}\sin(10\pi x_1)\right]$ $g(\vec{x}) = 1 + 9(\sum_{i=2}^{n} x_i)/(n-1)$	$0 \leq x_i \leq 1$	30
ZDT4	$f_1(\vec{x}) = x_1$ $f_2(\vec{x}) = g(\vec{x})[1-(x_1/g(\vec{x}))^2]$ $g(\vec{x}) = 1 + 10(n-1) + \sum_{i=2}^{n}[x_i^2 - 10\cos(4\pi x_i)]$	$0 \leq x_1 \leq 1$ $-5 \leq x_i \leq 5$ $i = 2, ..., n$	10
ZDT6	$f_1(\vec{x}) = 1 - e^{-4x_1}\sin^6(6\pi x_1)$ $f_2(\vec{x}) = g(\vec{x})[1-(f_1(\vec{x})/g(\vec{x}))^2]$ $g(\vec{x}) = 1 + 9[(\sum_{i=2}^{n} x_i)/(n-1)]^{0.25}$	$0 \leq x_i \leq 1$	10

Tabla A.4: Funciones de prueba con restricciones.

Problema	Funciones objetivo	Restricciones	Rangos de las variables	n
Osyczka2	$f_1(\vec{x}) = -(25(x_1-2)^2 + (x_2-2)^2 + (x_3-1)^2(x_4-4)^2 + (x_5-1)^2)$ $f_2(\vec{x}) = x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 + x_6^2$	$g_1(\vec{x}) = 0 \leq x_1 + x_2 - 2$ $g_2(\vec{x}) = 0 \leq 6 - x_1 - x_2$ $g_3(\vec{x}) = 0 \leq 2 - x_2 + x_1$ $g_4(\vec{x}) = 0 \leq 2 - x_1 + 3x_2$ $g_5(\vec{x}) = 0 \leq 4 - (x_3-3)^2 - x_4$ $g_6(\vec{x}) = 0 \leq (x_5-3)^3 + x_6 - 4$	$0 \leq x_1, x_2 \leq 10$ $1 \leq x_3, x_5 \leq 5$ $0 \leq x_4 \leq 6$ $0 \leq x_6 \leq 10$	6
Tanaka	$f_1(\vec{x}) = x_1$ $f_2(\vec{x}) = x_2$	$g_1(\vec{x}) = -x_1^2 - x_2^2 + 1 + 0.1\cos(16\arctan(x_1/x_2)) \leq 0$ $g_2(\vec{x}) = (x_1-0.5)^2 + (x_2-0.5)^2 \leq 0.5$	$-\pi \leq x_i \leq \pi$	2
ConstrEx	$f_1(\vec{x}) = x_1$ $f_2(\vec{x}) = (1+x_2)/x_1$	$g_1(\vec{x}) = x_2 + 9x_1 \geq 6$ $g_2(\vec{x}) = -x_2 + 9x_1 \geq 1$	$0.1 \leq x_1 \leq 1.0$ $0 \leq x_2 \leq 5$	2
Srinivas	$f_1(\vec{x}) = (x_1-2)^2 + (x_2-1)^2 + 2$ $f_2(\vec{x}) = 9x_1 - (x_2-1)^2$	$g_1(\vec{x}) = x_1^2 + x_2^2 \leq 225$ $g_2(\vec{x}) = x_1 - 3x_2 \leq -10$	$-20 \leq x_i \leq 20$	2

Tabla A.5: Propiedades de los MOPs creados con WFG.

Problema	Separabilidad	Modalidad	Sesgo	Geometría
WFG1	separable	uni	polinómico, plano	convexa, mezclada
WFG2	no separable	f_1 uni, f_2 multi	no tiene	convexa, desconectada
WFG3	no separable	uni	no tiene	lineal, degenerada
WFG4	no separable	multi	no tiene	cóncava
WFG5	separable	deceptiva	no tiene	cóncava
WFG6	no separable	uni	no tiene	cóncava
WFG7	separable	uni	dependiente del parámetro	cóncava
WFG8	no separable	uni	dependiente del parámetro	cóncava
WFG9	no separable	multi, deceptiva	dependiente del parámetro	cóncava

A.4. Conclusiones

Se ha presentado en este capítulo un elevado número de problemas que se han utilizado en los estudios desarrollados a lo largo de esta tesis, con la excepción de los problemas tomados del mundo real: VRP, DFCNT y cDFCNT; que son abordados en capítulos independientes.

Se presentan este capítulo problemas pertenecientes a los dominios de la optimización combinatoria, numérica y multi-objetivo. Todos los problemas propuestos son bien conocidos en la literatura, y ampliamente utilizados en los estudios pertenecientes a sus respectivos dominios. Además, se ha procurado elegir dentro de cada dominio un conjunto de problemas suficientemente amplio como para incluir un elevado número de características diversas, como la multimodalidad, epistasis, generadores de problemas, o problemas engañosos, que son especialmente diseñados para ser difíciles de resolver por algoritmos evolutivos.

Apéndice B

Resultados y Mejores Soluciones Encontradas para VRP

B.1. Mejores Soluciones Encontradas

En esta sección incluimos todos los detalles de las nuevas soluciones encontradas por JCell2o1i para futuros estudios. Los resultados mejorados pertenecen al conjunto de problemas de Van Breedam (tablas B.1 a B.8), y Taillard (Tabla B.9). En estas tablas, mostramos el número de rutas que componen la solución, su longitud global (suma de las distancias de cada ruta), y la composición de cada ruta (capacidad del vehículo utilizado, distancia de las rutas y el orden de visita de los clientes). Note que, debido a la representación del problema que utilizamos (véase la Sección 13.2.1), los clientes se numeran del 0 al $c - 1$ (0 para el primer cliente, no para el almacén), siendo c el número de clientes para la instancia en cuestión. El apéndice se incluye para hacer que el trabajo sea autocontenido y significativo para futuras extensiones y comparaciones.

Tabla B.1: Nueva solución encontrada para la instancia `Bre-1`.

Número de rutas		10
Longitud total de la solución		1106.0
Ruta 1	Capacidad	100
	Distancia	121.0
	Clientes visitados	69, 89, 99, 34, 39, 44, 49, 90, 70, 50
Ruta 2	Capacidad	100
	Distancia	98.0
	Clientes visitados	1, 6, 11, 64, 84, 26, 21, 16, 73, 53
Ruta 3	Capacidad	100
	Distancia	96.0
	Clientes visitados	0, 5, 10, 62, 82, 25, 20, 15, 71, 51
Ruta 4	Capacidad	100
	Distancia	124.0
	Clientes visitados	56, 76, 96, 47, 42, 37, 32, 95, 85, 65
Ruta 5	Capacidad	100
	Distancia	101.0
	Clientes visitados	2, 7, 12, 66, 86, 27, 22, 17, 75, 55
Ruta 6	Capacidad	100
	Distancia	122.0
	Clientes visitados	58, 78, 98, 48, 43, 38, 33, 97, 87, 67
Ruta 7	Capacidad	100
	Distancia	98.0
	Clientes visitados	4, 9, 14, 60, 80, 29, 24, 19, 79, 59
Ruta 8	Capacidad	100
	Distancia	123.0
	Clientes visitados	54, 74, 94, 46, 41, 36, 31, 93, 83, 63
Ruta 9	Capacidad	100
	Distancia	124.0
	Clientes visitados	52, 72, 92, 45, 40, 35, 30, 91, 81, 61
Ruta 10	Capacidad	100
	Distancia	99.0
	Clientes visitados	57, 77, 18, 23, 28, 88, 68, 13, 8, 3

Tabla B.2: Nueva mejor solución para la instancia `Bre-2`.

Número de rutas		10
Longitud total de la solución		1506.0
Ruta 1	Capacidad	100
	Distancia	121.0
	Clientes visitados	97, 87, 77, 67, 57, 56, 66, 76, 86, 96
Ruta 2	Capacidad	100
	Distancia	121.0
	Clientes visitados	93, 83, 73, 63, 53, 52, 62, 72, 82, 92
Ruta 3	Capacidad	100
	Distancia	184.0
	Vlientes visitados	48, 38, 28, 18, 8, 9, 19, 29, 39, 49
Ruta 4	Capacidad	100
	Distancia	149.0
	Clientes visitados	91, 81, 71, 61, 51, 50, 60, 70, 80, 90
Ruta 5	Capacidad	100
	Distancia	184.0
	Clientes visitados	41, 31, 21, 11, 1, 0, 10, 20, 30, 40
Ruta 6	Capacidad	100
	Distancia	158.0
	Clientes visitados	45, 35, 25, 15, 5, 4, 14, 24, 34, 44
Ruta 7	Capacidad	100
	Distancia	108.0
	Clientes visitados	94, 84, 74, 64, 54, 55, 65, 75, 85, 95
Ruta 8	Capacidad	100
	Distancia	149.0
	Clientes visitados	99, 89, 79, 69, 59, 58, 68, 78, 88, 98
Ruta 9	Capacidad	100
	Distancia	166.0
	Clientes visitados	47, 37, 27, 17, 7, 6, 16, 26, 36, 46
Ruta 10	Capacidad	100
	Distancia	166.0
	Clientes visitados	43, 33, 23, 13, 3, 2, 12, 22, 32, 42

Tabla B.3: Nueva mejor solución para la instancia `Bre-4`.

Número de rutas		20
Longitud total de la solución		1470.00
Ruta 1	Capacidad	50
	Distancia	82.0
	Clientes visitados	46, 47, 48, 49, 41
Ruta 2	Capacidad	50
	Distancia	51.0
	Clientes visitados	69, 66, 61, 62, 67
Ruta 3	Capacidad	50
	Distancia	60.0
	Clientes visitados	99, 98, 97, 96, 90
Ruta 4	Capacidad	50
	Distancia	74.0
	Clientes visitados	78, 81, 76, 74, 71
Ruta 5	Capacidad	50
	Distancia	111.0
	Clientes visitados	88, 95, 93, 86, 82
Ruta 6	Capacidad	50
	Distancia	39.0
	Clientes visitados	83, 91, 92, 84, 73
Ruta 7	Capacidad	50
	Distancia	111.0
	Clientes visitados	20, 14, 10, 5, 12
Ruta 8	Capacidad	50
	Distancia	82.0
	Clientes visitados	58, 50, 51, 52, 53
Ruta 9	Capacidad	50
	Distancia	74.0
	Clientes visitados	29, 22, 19, 24, 27
Ruta 10	Capacidad	50
	Distancia	51.0
	Clientes visitados	64, 59, 60, 65, 68
Ruta 11	Capacidad	50
	Distancia	51.0
	Clientes visitados	35, 40, 39, 34, 31
Ruta 12	Capacidad	50
	Distancia	111.0
	Clientes visitados	11, 4, 6, 13, 17
Ruta 13	Capacidad	50
	Distancia	111.0
	Clientes visitados	87, 94, 89, 85, 79
Ruta 14	Capacidad	50
	Distancia	74.0
	Clientes visitados	28, 21, 18, 23, 25
Ruta 15	Capacidad	50
	Distancia	82.0
	Clientes visitados	54, 55, 56, 57, 63
Ruta 16	Capacidad	50
	Distancia	74.0
	Clientes visitados	70, 77, 80, 75, 72
Ruta 17	Capacidad	50
	Distancia	60.0
	Clientes visitados	9, 3, 2, 1, 0
Ruta 18	Capacidad	50
	Distancia	39.0
	Clientes visitados	26, 15, 7, 8, 16
Ruta 19	Capacidad	50
	Distancia	82.0
	Clientes visitados	36, 42, 43, 44, 45
Ruta 20	Capacidad	50
	Distancia	51.0
	Clientes visitados	30, 33, 38, 37, 32

Tabla B.4: Nueva mejor solución para la instancia `Bre-5`.

Número de rutas		5
Longitud total de la solución		950.0
Ruta 1	Capacidad	200
	Distancia	180.0
	Clientes visitados	97, 88, 98, 78, 89, 69, 99, 79, 59, 40, 30, 20, 39, 29, 49, 38, 28, 48, 57, 77
Ruta 2	Capacidad	200
	Distancia	228.0
	Clientes visitados	45, 34, 44, 63, 83, 93, 73, 53, 64, 84, 94, 74, 54, 65, 85, 95, 75, 55, 66, 86
Ruta 3	Capacidad	200
	Distancia	135.0
	Clientes visitados	96, 76, 56, 46, 35, 25, 14, 24, 13, 23, 12, 3, 4, 5, 15, 6, 16, 36, 67, 87
Ruta 4	Capacidad	200
	Distancia	153.0
	Clientes visitados	37, 17, 8, 18, 9, 19, 0, 10, 21, 41, 31, 42, 32, 22, 11, 2, 1, 7, 27, 47
Ruta 5	Capacidad	200
	Distancia	254.0
	Clientes visitados	26, 33, 43, 52, 72, 92, 82, 62, 51, 71, 91, 81, 61, 50, 70, 90, 80, 60, 58, 68

Tabla B.5: Nueva mejor solución para la instancia `Bre-6`.

Número de rutas		5
Longitud total de la solución		969.0
Ruta 1	Capacidad	200
	Distancia	151.0
	Clientes visitados	97, 96, 37, 30, 70, 23, 16, 64, 9, 65, 68, 49, 73, 31, 83, 38, 90, 91, 45, 98
Ruta 2	Capacidad	200
	Distancia	217.0
	Clientes visitados	78, 74, 19, 12, 5, 54, 55, 56, 57, 6, 13, 20, 27, 34, 41, 82, 86, 93, 95, 48
Ruta 3	Capacidad	200
	Distancia	218.0
	Clientes visitados	42, 94, 89, 85, 79, 35, 28, 21, 14, 7, 0, 50, 51, 52, 53, 1, 8, 72, 77, 44
Ruta 4	Capacidad	200
	Distancia	196.0
	Clientes visitados	99, 92, 84, 24, 69, 17, 10, 3, 2, 60, 59, 58, 15, 22, 29, 75, 80, 87, 36, 43
Ruta 5	Capacidad	200
	Distancia	187.0
	Clientes visitados	46, 39, 32, 71, 25, 18, 66, 67, 11, 61, 4, 62, 63, 26, 33, 76, 81, 88, 40, 47

Tabla B.6: Nueva mejor solución para la instancia `Bre-9`.

Número de rutas		10
Longitud total de la solución		1690.0
Ruta 1	Capacidad	100
	Distancia	143.0
	Clientes visitados	47, 48, 41, 90, 88, 84, 82, 34, 86, 40, 46
Ruta 2	Capacidad	100
	Distancia	146.0
	Clientes visitados	32, 78, 73, 26, 27, 75, 81, 33, 39
Ruta 3	Capacidad	100
	Distancia	208.0
	Clientes visitados	72, 22, 15, 8, 58, 59, 60, 2, 9, 65, 68
Ruta 4	Capacidad	100
	Distancia	232.0
	Clientes visitados	20, 13, 6, 57, 56, 55, 54, 5, 11, 25, 92
Ruta 5	Capacidad	100
	Distancia	64.0
	Clientes visitados	97, 45, 94, 91, 38, 95, 99, 98
Ruta 6	Capacidad	100
	Distancia	146.0
	Clientes visitados	96, 37, 30, 70, 23, 77, 29, 74, 80, 85, 36, 44
Ruta 7	Capacidad	100
	Distancia	140.0
	Clientes visitados	79, 83, 87, 89, 35, 42, 43
Ruta 8	Capacidad	100
	Distancia	234.0
	Clientes visitados	16, 64, 1, 53, 52, 51, 50, 0, 7, 14, 21, 28
Ruta 9	Capacidad	100
	Distancia	157.0
	Clientes visitados	93, 31, 76, 49, 69, 66, 67, 18, 71
Ruta 10	Capacidad	100
	Distancia	220.0
	Clientes visitados	19, 12, 63, 62, 61, 4, 3, 10, 17, 24

Tabla B.7: Nueva mejor solución para la instancia `Bre-10`.

Número de rutas		10
Longitud total de la solución		1026.0
Ruta 1	Capacidad	100
	Distancia	80.0
	Clientes visitados	83, 81, 67, 63, 65, 64, 69, 80, 82, 1
Ruta 2	Capacidad	100
	Distancia	63.0
	Clientes visitados	6, 2, 27, 23, 20, 22, 24, 26, 28, 29
Ruta 3	Capacidad	100
	Distancia	135.0
	Clientes visitados	35, 34, 31, 30, 32, 36, 38, 39, 13
Ruta 4	Capacidad	100
	Distancia	135.0
	Clientes visitados	49, 44, 41, 58, 54, 59, 60, 61, 62, 66, 68
Ruta 5	Capacidad	100
	Distancia	206.0
	Clientes visitados	37, 33, 56, 52, 50, 51, 53, 55, 57, 70, 72
Ruta 6	Capacidad	100
	Distancia	68.0
	Clientes visitados	17, 19, 18, 16, 14, 15, 12, 10, 11
Ruta 7	Capacidad	100
	Distancia	55.0
	Clientes visitados	7, 88, 89, 87, 85, 84, 86, 3, 5, 9
Ruta 8	Capacidad	100
	Distancia	136.0
	Clientes visitados	93, 78, 79, 77, 75, 73, 71, 74, 76
Ruta 9	Capacidad	100
	Distancia	67.0
	Clientes visitados	92, 90, 91, 94, 95, 97, 99, 98, 96
Ruta 10	Capacidad	100
	Distancia	81.0
	Clientes visitados	8, 4, 0, 47, 45, 43, 40, 42, 46, 48, 21, 25

Tabla B.8: Nueva mejor solución para la instancia `Bre-11`.

Número de rutas		10
Longitud total de la solución		1128.0
Ruta 1	Capacidad	100
	Distancia	117.0
	Clientes visitados	36, 26, 16, 6, 7, 17, 27, 46, 56, 66
Ruta 2	Capacidad	100
	Distancia	105.0
	Clientes visitados	93, 82, 91, 90, 80, 70, 71, 72, 83
Ruta 3	Capacidad	100
	Distancia	60.0
	Clientes visitados	95, 85, 75, 55, 54, 64, 74, 84, 94
Ruta 4	Capacidad	100
	Distancia	105.0
	Clientes visitados	73, 62, 61, 51, 41, 31, 42, 52, 63
Ruta 5	Capacidad	100
	Distancia	107.0
	Clientes visitados	86, 77, 58, 48, 38, 37, 47, 57, 67, 76
Ruta 6	Capacidad	100
	Distancia	152.0
	Clientes visitados	32, 21, 11, 1, 0, 10, 20, 30, 40, 50, 60, 81, 92
Ruta 7	Capacidad	100
	Distancia	105.0
	Clientes visitados	44, 34, 24, 14, 4, 5, 15, 25, 35, 45, 65
Ruta 8	Capacidad	100
	Distancia	116.0
	Clientes visitados	43, 33, 23, 13, 3, 2, 12, 22, 53
Ruta 9	Capacidad	100
	Distancia	112.0
	Clientes visitados	87, 88, 78, 68, 79, 89, 99, 98, 97, 96
Ruta 10	Capacidad	100
	Distancia	149.0
	Clientes visitados	69, 59, 49, 39, 29, 19, 9, 8, 18, 28

Tabla B.9: Nueva mejor solución para la instancia `ta-n76-k10b`.

Número de rutas		10
Longitud total de la solución		1344.6186241386285
Ruta 1	Capacidad	1654
	Distancia	258.5993825897365
	Clientes visitados	73, 9, 11, 68, 54, 55, 60, 57, 56, 59, 61, 58, 67, 69, 35
Ruta 2	Capacidad	1659
	Distancia	174.46520984752948
	Clientes visitados	39, 22, 15, 17, 31, 46, 45, 13
Ruta 3	Capacidad	1590
	Distancia	72.761072402546
	Clientes visitados	12, 42, 49, 47
Ruta 4	Capacidad	1594
	Distancia	321.3083147478987
	Clientes visitados	7, 72, 62, 65, 66, 63, 64, 52
Ruta 5	Capacidad	1219
	Distancia	13.508270432752166
	Clientes visitados	6, 10, 0
Ruta 6	Capacidad	1677
	Distancia	214.36282712025178
	Clientes visitados	25, 33, 23, 16, 30, 27, 34, 24, 28, 19, 26, 20, 21, 29, 32
Ruta 7	Capacidad	1617
	Distancia	90.52827230772104
	Clientes visitados	50, 38, 41, 53, 48, 36, 43, 5
Ruta 8	Capacidad	1547
	Distancia	32.385178886685395
	Clientes visitados	1, 4, 74, 71, 14, 8
Ruta 9	Capacidad	687
	Distancia	6.0
	Clientes visitados	3
Ruta 10	Capacidad	1662
	Distancia	160.7000958035075
	Clientes visitados	2, 51, 37, 18, 44, 40, 70

B.2. Detalles Numéricos Exhaustivos para VRP

Las tablas que contienen los resultados obtenidos en nuestros estudios con JCell2o1i al resolver todas las instancias propuestas en este trabajo se muestran en esta sección. Los valores incluidos en las Tablas B.10 a B.19 se han obtenido realizando 100 ejecuciones independientes (para obtener resultados significativos), excepto para el conjunto de problemas de Golden et al. (Tabla B.17), donde sólo se realizan 30 ejecuciones debido a su dificultad.

Los valores de las tablas de esta sección se corresponden con la mejor solución encontrada en nuestras ejecuciones para cada instancia, el número medio de evaluaciones realizadas para encontrar dicha solución, la tasa de éxito, el valor medio de soluciones encontradas en todas las ejecuciones independientes, la desviación (Δ) entre nuestra mejor solución y la mejor conocida hasta el momento (en tanto por ciento) y la mejor solución conocida para cada instancia. Los tiempos de ejecución de nuestras pruebas van desde 153.5 horas para la instancia más compleja a 0.32 segundos para la más sencilla.

Tabla B.10: Resultados computacionales para los problemas de Augerat et al. Set A.

Instancia	**Mejor Solución**	**Mejor Solución Conocida Encontrada**		**Solución media**	**Δ**	**Mejor Conocida**
		Evaluaciones Media	**Éxito (%)**		**(%)**	**Previamente**
A-n32-k5	784.00	10859.36 ± 3754.35	100.00	784.00 ± 0.00	0.00	784.0 [109]
A-n33-k5	661.00	10524.08 ± 2773.86	100.00	661.00 ± 0.00	0.00	661.0 [109]
A-n33-k6	742.00	9044.89 ± 3087.92	100.00	742.00 ± 0.00	0.00	742.0 [109]
A-n34-k5	778.00	13172.95 ± 4095.90	100.00	778.00 ± 0.00	0.00	778.0 [109]
A-n36-k5	799.00	21699.29 ± 5927.13	86.00	800.39 ± 3.62	0.00	799.0 [109]
A-n37-k5	669.00	15101.04 ± 4227.04	100.00	669.00 ± 0.00	0.00	669.0 [109]
A-n37-k6	949.00	20575.88 ± 6226.19	95.00	949.12 ± 0.67	0.00	949.0 [109]
A-n38-k5	730.00	17513.31 ± 5943.05	96.00	730.07 ± 0.36	0.00	730.0 [109]
A-n39-k5	822.00	25332.52 ± 7269.13	21.00	824.77 ± 1.71	0.00	822.0 [109]
A-n39-k6	831.00	22947.67 ± 4153.08	12.00	833.16 ± 1.35	0.00	831.0 [109]
A-n44-k7	937.00	39208.71 ± 11841.16	34.00	939.53 ± 2.69	0.00	937.0 [109]
A-n45-k6	944.00	52634.00 ± 10484.88	6.00	957.88 ± 7.85	0.00	944.0 [109]
A-n45-k7	1146.00	46516.30 ± 13193.60	76.00	1147.17 ± 2.23	0.00	1146.0 [109]
A-n46-k7	914.00	32952.40 ± 8327.32	100.00	914.00 ± 0.00	0.00	914.0 [109]
A-n48-k7	1073.00	41234.69 ± 11076.23	55.00	1078.68 ± 6.89	0.00	1073.0 [109]
A-n53-k7	1010.00	56302.26 ± 15296.18	19.00	1016.10 ± 4.17	0.00	1010.0 [109]
A-n54-k7	1167.00	58062.13 ± 12520.94	52.00	1170.68 ± 4.84	0.00	1167.0 [109]
A-n55-k9	1073.00	50973.67 ± 10628.18	48.00	1073.87 ± 2.15	0.00	1073.0 [109]
A-n60-k9	1354.00	97131.75 ± 13568.88	8.00	1359.82 ± 4.53	0.00	1354.0 [109]
A-n61-k9	1034.00	87642.33 ± 8468.60	6.00	1040.50 ± 7.43	0.00	1034.0 [109]
A-n62-k8	1288.00	166265.86 ± 27672.54	7.00	1300.92 ± 7.12	0.00	1288.0 [109]
A-n63-k9	1616.00	131902.00 ± 12010.92	2.00	1630.59 ± 5.65	0.00	1616.0 [109]
A-n63-k10	1314.00	90994.00 ± 0.0	1.00	1320.97 ± 3.69	0.00	1314.0 [109]
A-n64-k9	1401.00	100446.00 ± 17063.90	2.00	1416.12 ± 6.38	0.00	1401.0 [109]
A-n65-k9	1174.00	88292.50 ± 20815.10	2.00	1181.60 ± 3.51	0.00	1174.0 [109]
A-n69-k9	1159.00	90258.00 ± 0.0	1.00	1168.56 ± 3.64	0.00	1159.0 [109]
A-n80-k10	1763.00	154976.00 ± 33517.14	4.00	1785.78 ± 10.88	0.00	1763.0 [109]

Tabla B.11: Resultados computacionales para los problemas de Augerat et al. Set B.

Instancia	Mejor Solución	Mejor Solución Conocida Encontrada		Solución media	Δ	Mejor Conocida
		Evaluaciones Media	Éxito (%)		(%)	Previamente
B-n31-k5	672.00	9692.02 ± 3932.91	100.00	672.00 ± 0.00	0.00	672.0 [109]
B-n34-k5	788.00	11146.94 ± 4039.59	100.00	788.00 ± 0.00	0.00	788.0 [109]
B-n35-k5	955.00	12959.81 ± 4316.64	100.00	955.00 ± 0.00	0.00	955.0 [109]
B-n38-k6	805.00	23227.87 ± 4969.47	92.00	805.08 ± 0.27	0.00	805.0 [109]
B-n39-k5	549.00	21540.88 ± 6813.76	85.00	549.20 ± 0.57	0.00	549.0 [109]
B-n41-k6	829.00	31148.89 ± 5638.03	79.00	829.21 ± 0.41	0.00	829.0 [109]
B-n43-k6	742.00	27894.05 ± 5097.48	60.00	742.86 ± 1.22	0.00	742.0 [109]
B-n44-k7	909.00	31732.34 ± 5402.31	85.00	910.34 ± 4.17	0.00	909.0 [109]
B-n45-k5	751.00	36634.68 ± 4430.40	28.00	754.52 ± 3.88	0.00	751.0 [109]
B-n45-k6	678.00	58428.21 ± 9709.70	33.00	680.94 ± 3.51	0.00	678.0 [109]
B-n50-k7	741.00	16455.25 ± 3159.24	100.00	741.00 ± 0.00	0.00	741.0 [109]
B-n50-k8	1312.00	59813.00 ± 0.0	1.00	1317.22 ± 4.32	0.00	1312.0 [109]
B-n51-k7	1032.00	59401.30 ± 12736.87	79.00	1032.26 ± 0.56	0.00	1032.0 [109]
B-n52-k7	747.00	32188.17 ± 5886.91	70.00	747.56 ± 1.18	0.00	747.0 [109]
B-n56-k7	707.00	45605.11 ± 7118.89	17.00	710.56 ± 2.22	0.00	707.0 [109]
B-n57-k7	1153.00	106824.57 ± 9896.32	7.00	1168.97 ± 15.52	0.00	1153.0 [109]
B-n57-k9	1598.00	71030.38 ± 7646.15	16.00	1601.64 ± 3.66	0.00	1598.0 [109]
B-n63-k10	1496.00	67313.50 ± 10540.84	2.00	1530.51 ± 14.07	0.00	1496.0 [109]
B-n64-k9	861.00	97043.32 ± 15433.00	19.00	866.69 ± 6.04	0.00	861.0 [109]
B-n66-k9	1316.00	94125.50 ± 15895.05	2.00	1321.36 ± 2.94	0.00	1316.0 [109]
B-n67-k10	1032.00	130628.60 ± 26747.01	5.00	1042.48 ± 11.63	0.00	1032.0 [109]
B-n68-k9	1273.00	—	0.00	1286.19 ± 3.88	0.08	1272.0 [109]
B-n78-k10	1221.00	145702.71 ± 16412.80	14.00	1239.86 ± 13.05	0.00	1221.0 [109]

Tabla B.12: Resultados computacionales para los problemas de Augerat et al. Set P.

Instancia	Mejor Solución	Mejor Solución Conocida Encontrada		Solución media	Δ	Mejor Conocida
		Evaluaciones Media	Éxito (%)		(%)	Previamente
P-n16-k8	450.00	1213.92 ± 325.60	100.00	450.00 ± 0.00	0.00	450.0 [109]
P-n19-k2	212.00	2338.79 ± 840.45	100.00	212.00 ± 0.00	0.00	212.0 [109]
P-n20-k2	216.00	3122.74 ± 1136.45	100.00	216.00 ± 0.00	0.00	216.0 [109]
P-n21-k2	211.00	1704.60 ± 278.63	100.00	211.00 ± 0.00	0.00	211.0 [109]
P-n22-k2	216.00	2347.77 ± 505.00	100.00	216.00 ± 0.00	0.00	216.0 [109]
P-n22-k8	603.00	2708.12 ± 877.52	100.00	603.00 ± 0.00	0.00	603.0 [109]
P-n23-k8	529.00	3422.29 ± 1275.23	100.00	529.00 ± 0.00	0.00	529.0 [109]
P-n40-k5	458.00	16460.94 ± 4492.84	99.00	458.01 ± 0.10	0.00	458.0 [109]
P-n45-k5	510.00	25434.43 ± 6480.60	72.00	510.89 ± 1.68	0.00	510.0 [109]
P-n50-k7	554.00	45280.10 ± 9004.78	29.00	556.08 ± 1.94	0.00	554.0 [109]
P-n50-k8	631.00	62514.50 ± 8506.60	4.00	638.80 ± 5.58	0.00	631.0 [109]
P-n50-k10	696.00	49350.00 ± 0.0	1.00	698.58 ± 2.65	0.00	696.0 [109]
P-n51-k10	741.00	55512.21 ± 9290.20	19.00	747.25 ± 6.65	0.00	741.0 [109]
P-n55-k7	568.00	57367.27 ± 11916.64	22.00	573.03 ± 3.09	0.00	568.0 [109]
P-n55-k8	576.00	48696.50 ± 12961.64	78.00	576.42 ± 1.15	0.00	576.0 [109]
P-n55-k10	694.00	47529.00 ± 0.0	1.00	698.22 ± 1.81	0.00	694.0 [109]
P-n55-k15	989.00	108412.75 ± 8508.23	4.00	1002.96 ± 9.82	0.00	989.0 [109]
P-n60-k10	744.00	77925.62 ± 12377.87	8.00	749.35 ± 4.95	0.00	744.0 [109]
P-n60-k15	968.00	87628.35 ± 20296.24	23.00	972.42 ± 3.12	0.00	968.0 [109]
P-n65-k10	792.00	66024.53 ± 12430.83	30.00	798.44 ± 5.03	0.00	792.0 [109]
P-n70-k10	827.00	131991.00 ± 11429.62	4.00	836.28 ± 5.70	0.00	827.0 [109]
P-n76-k4	593.00	89428.14 ± 17586.60	7.00	599.73 ± 5.07	0.00	593.0 [109]
P-n76-k5	627.00	150548.60 ± 38449.91	5.00	632.89 ± 4.51	0.00	627.0 [109]
P-n101-k4	681.00	107245.00 ± 26834.63	7.00	687.36 ± 5.22	0.00	681.0 [109]

Tabla B.13: Resultados computacionales para los problemas de Van Breedam.

Instancia	Mejor Solución	Mejor Solución Conocida Encontrada		Solución media	Δ	Mejor Conocida
		Evaluaciones Media	Éxito (%)		(%)	Previamente
Bre-1	**1106.00**	428896.92 ± 365958.66	65.00	1113.03 ± 10.80	3.24	1143.0 [281]
Bre-2	**1506.00**	365154.72 ± 196112.12	53.00	1513.68 ± 11.07	4.68	1580.0 [281]
Bre-3	1751.00	146429.00 ± 29379.40	100.00	1751.00 ± 0.00	0.00	1751.0 [281]
Bre-4	**1470.00**	210676.00 ± 82554.15	100.00	1470.00 ± 0.00	0.41	1476.0 [281]
Bre-5	**950.00**	977540.00 ± 554430.07	5.00	955.36 ± 4.16	2.56	975.0 [281]
Bre-6	**969.00**	820180.39 ± 436314.36	51.00	970.79 ± 2.46	1.02	979.0 [281]
Bre-9	**1690.00**	1566300.00 ± 0.0	1.00	1708.48 ± 10.48	5.59	1790.0 [281]
Bre-10	**1026.00**	1126575.00 ± 428358.44	4.00	1033.34 ± 6.22	1.82	1045.0 [281]
Bre-11	**1128.00**	1742600.00 ± 840749.96	2.00	1153.35 ± 10.81	2.76	1160.0 [281]

Tabla B.14: Resultados computacionales para los problemas de Christofides y Eilon.

Instancia	Mejor Solución	Mejor Solución Conocida Encontrada		Solución media	Δ	Mejor Conocida
		Evaluaciones Media	Éxito (%)		(%)	Previamente
E-n13-k4	247.00	4200.00 ± 0.00	100.00	247.00 ± 0.00	0.00	247.0 [109]
E-n22-k4	375.00	3831.60 ± 1497.55	100.00	375.00 ± 0.00	0.00	375.0 [109]
E-n23-k3	569.00	2068.78 ± 414.81	100.00	569.00 ± 0.00	0.00	569.0 [109]
E-n30-k3	534.00	6400.86 ± 2844.69	99.00	534.03 ± 0.30	0.00	534.0 [109]
E-n30-k4	503.00	4644.87 ± 1256.38	100.00	503.00 ± 0.00	—	—
E-n31-k7	379.00	274394, 03 ± 162620, 36	67.00	380.67 ± 3.36	0.00	379.0 [109]
E-n33-k4	835.00	14634.89 ± 4463.32	100.00	835.00 ± 0.00	0.00	835.0 [109]
E-n51-k5	521.00	40197.43 ± 10576.82	30.00	526.54 ± 4.75	0.00	521.0 [109]
E-n76-k7	682.00	80594.00 ± 20025.23	3.00	688.19 ± 3.42	0.00	682.0 [109]
E-n76-k8	735.00	86700.33 ± 26512.46	3.00	741.25 ± 4.14	0.00	735.0 [109]
E-n76-k10	830.00	166568.33 ± 11138.72	3.00	841.09 ± 6.16	0.00	830.0 [109]
E-n76-k14	1021.00	235643.50 ± 55012.20	2.00	1033.05 ± 6.14	0.00	1021.0 [109]
E-n101-k8	817.00	192223.62 ± 43205.58	8.00	822.84 ± 3.92	0.00	815.0 [109]
E-n101-k14	1071.00	1832800.00 ± 0.00	1.00	1089.60 ± 7.59	0.00	1071.0 [109]

Tabla B.15: Resultados computacionales para los problemas de Christofides, Mingozzi y Toth.

Instancia	Mejor Solución	Mejor Solución Conocida Encontrada		Solución media	Δ	Mejor Conocida
		Evaluaciones Media	Éxito (%)		(%)	Previamente
CMT-n51-k5	524.61	27778.16 ± 6408.68	51.00	525.70 ± 2.31	0.00	524.61 [113]
CMT-n76-k10	835.26	118140.50 ± 65108.27	2.00	842.90 ± 5.49	0.00	835.26 [266]
CMT-n101-k8	826.14	146887.00 ± 42936.94	2.00	833.21 ± 4.22	0.00	826.14 [266]
CMT-n101-k10	819.56	70820.85 ± 14411.87	89.00	821.01 ± 4.89	0.00	819.56 [266]
CMT-n121-k7	1042.12	352548.00 ± 0.00	1.00	1136.95 ± 31.36	0.00	1042.11 [266]
CMT-n151-k12	1035.22	—	0.00	1051.74 ± 7.27	0.66	1028.42 [266]
CMT-n200-k17	1315.67	—	0.00	1339.90 ± 11.30	1.88	1291.45 [200]
CMTd-n51-k6	555.43	27413.30 ± 6431.00	23.00	558.10 ± 2.15	0.00	555.43 [266]
CMTd-n76-k11	909.68	69638.50 ± 12818.35	6.00	919.30 ± 7.22	0.00	909.68 [266]
CMTd-n101-k9	865.94	91882.00 ± 0.00	1.00	877.01 ± 5.92	0.00	865.94 [266]
CMTd-n101-k11	866.37	87576.85 ± 13436.17	68.00	867.53 ± 3.65	0.00	866.37 [226]
CMTd-n121-k11	1543.63	—	0.00	1568.53 ± 19.57	0.16	1541.14 [266]
CMTd-n151-k14	1165.10	—	0.00	1182.42 ± 10.50	0.22	1162.55 [266]
CMTd-n200-k18	1410.92	—	0.00	1432.94 ± 11.41	1.08	1395.85 [242]

Tabla B.16: Resultados computacionales para los problemas de Fisher.

Instancia	Mejor Solución	Mejor Solución Conocida Encontrada		Solución media	Δ	Mejor Conocida
		Evaluaciones Media	Éxito (%)		(%)	Previamente
F-n45-k4	724.00	14537.45 ± 4546.13	96.00	724.16 ± 0.93	0.00	724.0 [109]
F-n72-k4	237.00	25572.12 ± 5614.46	84.00	237.30 ± 0.83	0.00	237.0 [109]
F-n135-k7	1162.00	167466.33 ± 30768.37	3.00	1184.76 ± 20.04	0.00	1162.0 [109]

Tabla B.17: Resultados computacionales para los problemas de Golden et al.

Instancia	Mejor Solución	Mejor Solución Conocida Encontrada		Solución media	Δ	Mejor Conocida
		Evaluaciones Media	Éxito (%)		(%)	Previamente
gol-n241-k22	714.73	—	0.00	724.96 ± 7.11	0.98	707.79 [200]
gol-n253-k27	872.59	—	0.00	884.05 ± 5.98	1.57	859.11 [200]
gol-n256-k14	589.20	—	0.00	601.10 ± 5.43	0.99	583.39 [200]
gol-n301-k28	1019.19	—	0.00	1037.31 ± 18.59	2.04	998.73 [200]
gol-n321-k30	1107.24	—	0.00	1122.32 ± 9.71	2.43	1081.31 [200]
gol-n324-k16	754.95	—	0.00	766.31 ± 6.29	1.74	742.03 [181]
gol-n361-k33	1391.27	—	0.00	1409.42 ± 7.74	1.79	1366.86 [200]
gol-n397-k34	1367.47	—	0.00	1388.93 ± 8.52	1.65	1345.23 [200]
gol-n400-k18	935.91	—	0.00	950.07 ± 8.57	1.90	918.45 [200]
gol-n421-k41	1867.30	—	0.00	1890.16 ± 12.28	2.53	1821.15 [200]
gol-n481-k38	1663.87	—	0.00	1679.55 ± 9.64	2.54	1622.69 [200]
gol-n484-k19	1126.38	—	0.00	1140.03 ± 7.11	1.73	1107.19 [200]
gold-n201-k5	6460.98	1146050.00 ± 89873.27	2.00	6567.35 ± 104.68	0.00	6460.98 [270]
gold-n241-k10	5669.82	—	0.00	5788.72 ± 44.79	0.75	5627.54 [200]
gold-n281-k8	8428.18	—	0.00	8576.61 ± 89.38	0.18	8412.88 [270]
gold-n321-k10	8488.29	—	0.00	8549.17 ± 47.61	0.48	8447.92 [270]
gold-n361-k9	10227.51	—	0.00	10369.83 ± 69.64	0.31	10195.56 [270]
gold-n401-k10	11164.11	—	0.00	11293.47 ± 69.95	1.16	11036.23 [270]
gold-n441-k11	11946.05	—	0.00	12035.17 ± 63.45	2.42	11663.55 [200]
gold-n481-k12	13846.94	—	0.00	14052.71 ± 140.07	1.63	13624.52 [270]

Tabla B.18: Resultados computacionales para los problemas de Taillard.

Instancia	Mejor Solución	Mejor Solución Conocida Encontrada		Solución media	Δ	Mejor Conocida
		Evaluaciones Media	Éxito (%)		(%)	Previamente
ta-n76-k10a	1618.36	107401.97 ± 51052.17	65.00	1619.79 ± 2.18	0.00	1618.36 [266]
ta-n76-k10b	**1344.62**	75143.36 ± 20848.50	11.00	1345.10 ± 0.63	0.00	1344.64 [266]
ta-n76-k9a	1291.01	116404.75 ± 60902.73	72.00	1294.81 ± 10.02	0.00	1291.01 [266]
ta-n76-k9b	1365.42	98309.19 ± 36107.58	94.00	1365.53 ± 0.88	0.00	1365.42 [266]
ta-n101-k10a	1411.66	—	0.00	1419.41 ± 4.05	0.39	1406.20 [110]
ta-n101-k10b	1584.20	—	0.00	1599.57 ± 4.69	0.19	1581.25 [200]
ta-n101-k11a	2047.90	—	0.00	2070.39 ± 9.39	0.32	2041.34 [110]
ta-n101-k11b	1940.36	—	0.00	1943.60 ± 4.68	0.02	1939.90 [110]
ta-n151-k14a	2364.08	—	0.00	2407.75 ± 25.57	0.95	2341.84 [266]
ta-n151-k14b	2654.69	—	0.00	2688.06 ± 14.12	0.35	2645.39 [110]
ta-n151-k15a	3056.41	—	0.00	3124.28 ± 51.65	0.04	3055.23 [266]
ta-n151-k15b	2732.75	—	0.00	2782.21 ± 28.61	2.87	2656.47 [266]
ta-n386-k47	24941.71	—	0.00	25450.87 ± 165.41	2.09	24431.44 [242]

Tabla B.19: Factores computacionales de instancias tomadas del TSP.

Instancia	Mejor Solución	Mejor Solución Conocida Encontrada		Solución media	Δ	Mejor Conocida
		Evaluaciones Media	Éxito (%)		(%)	Previamente
att-n48-k4	40002.00	121267.06 ± 69462.74	85.00	40030.28 ± 69.04	0.00	40002.0 [1]
bayg-n29-k4	2050.00	32326.00 ± 20669.26	100.00	2050.00 ± 0.00	0.00	2050.0 [1]
bays-n29-k5	2963.00	18345.00 ± 6232.28	100.00	2963.00 ± 0.00	0.00	2963.0 [1]
dantzig-n42-k4	1142.00	71399.00 ± 45792.68	100.00	1142.00 ± 0.00	0.00	1142.0 [1]
fri-n26-k3	1353.00	12728.00 ± 6130.78	100.00	1353.00 ± 0.00	0.00	1353.0 [1]
gr-n17-k3	2685.00	4200.00 ± 0.00	100.00	2685.00 ± 0.00	0.00	2685.0 [1]
gr-n21-k3	3704.00	5881.00 ± 3249.57	100.00	3704.00 ± 0.00	0.00	3704.0 [1]
gr-n24-k4	2053.00	13835.00 ± 6313.48	100.00	2053.00 ± 0.00	0.00	2053.0 [1]
gr-n48-k3	5985.00	175558.97 ± 123635.58	39.00	5986.96 ± 1.75	0.00	5985.0 [1]
hk-n48-k4	14749.00	117319.00 ± 71898.28	100.00	14749.00 ± 0.00	0.00	14749.0 [1]
swiss-n42-k5	1668.00	64962.00 ± 28161.78	100.00	1668.00 ± 0.00	0.00	1668.0 [1]
ulysses-n16-k3	7965.00	4200.00 ± 0.00	100.00	7965.00 ± 0.00	0.00	7965.0 [1]
ulysses-n22-k4	9179.00	7603.00 ± 4401.78	100.00	9179.00 ± 0.00	0.00	9179.0 [1]

Apéndice C

Relación de Publicaciones que Sustentan esta Tesis Doctoral

En este apéndice se presenta el conjunto de trabajos que han sido publicados como fruto de las investigaciones desarrolladas a lo largo de este trabajo.

Estas publicaciones avalan el interés, la validez, y las aportaciones de esta tesis doctoral en la literatura, ya que estos trabajos han aparecido publicados en foros de prestigio y, por tanto, han sido sometidos a procesos de revisión por reconocidos investigadores especializados. En la Figura C.1 se muestra un esquema de las principales publicaciones que han aparecido en diversos congresos, libros, y revistas de prestigio internacional como fruto del trabajo de esta tesis doctoral. Además, está en proceso de elaboración un nuevo libro que permita dar a conocer a la comunidad científica internacional, entre otras cosas, la información recolectada en este trabajo. El libro *[1]* aparecerá publicado por una importante editorial científica (Springer-Verlag) a finales de 2007.

A continuación se muestran las referencias de todas las publicaciones en las que ha participado el autor de esta tesis doctoral. Un resumen de las mismas se muestra en la Tabla C.1.

Tabla C.1: Resumen de publicaciones.

Libros	1
Capítulos de Libro	6
Revistas Internacionales ISI	4
Revistas Internacionales	1
Publicaciones LNCS	5
Congresos Internacionales	8
Congresos Nacionales	3
Informes Técnicos	2

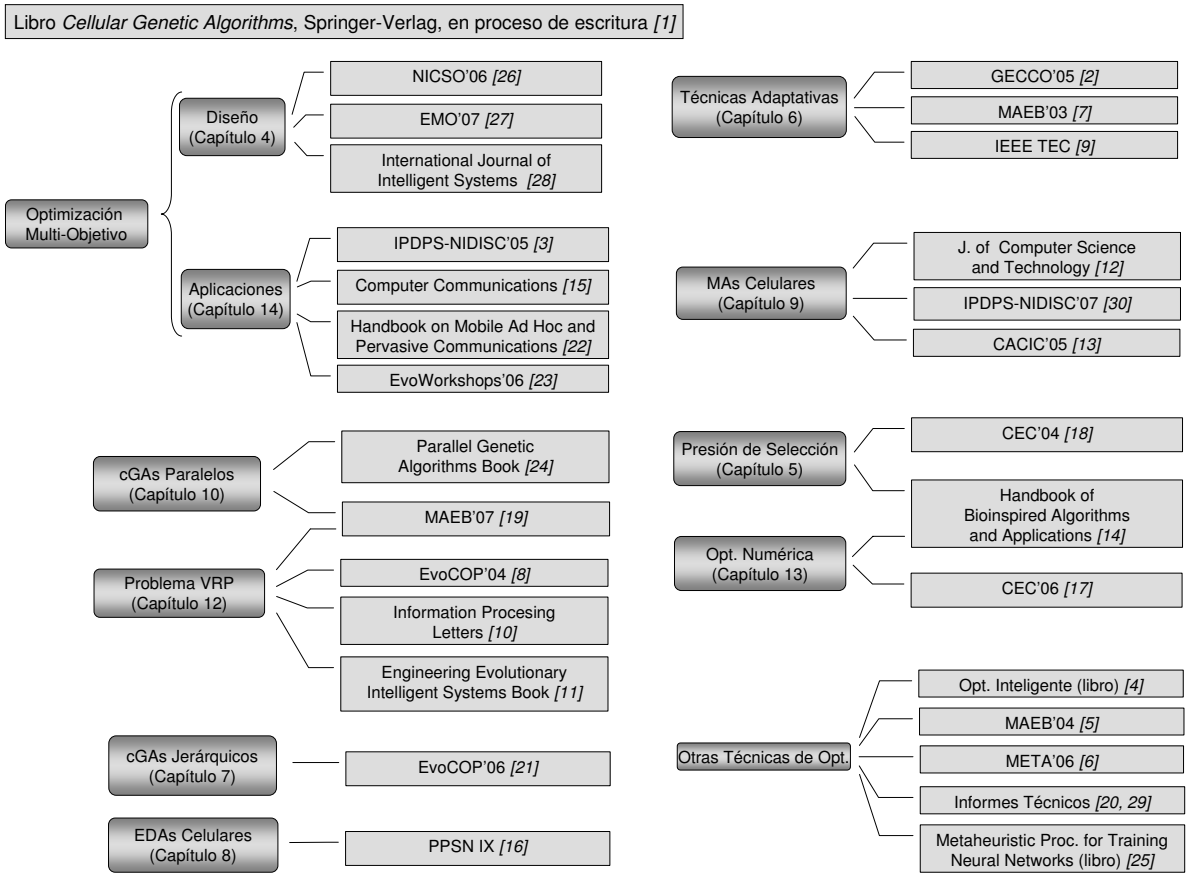


Figura C.1: Esquema de las publicaciones que avalan el trabajo realizado en esta tesis doctoral.

[1] E. Alba y B. Dorronsoro. Cellular Genetic Algorithms. Springer-Verlag, 2007, en proceso de escritura.

[2] E. Alba, H. Alfonso, y B. Dorronsoro. Advanced models of cellular genetic algorithms evaluated on SAT. En *Proc. of the Genetic and Evolutionary Computation COnference (GECCO)*, páginas 1123–1130, Washington D.C. USA, Junio de 2005. ACM Press.

[3] E. Alba, P. Bouvry, B. Dorronsoro, F. Luna, y A.J. Nebro. A cellular multi-objective genetic algorithm for optimal broadcasting strategy in metropolitan MANETs. In *Nature Inspired Distributed Computing (NIDISC) sessions of the International Parallel and Distributed Processing Simposium (IPDPS) 2005 Workshop*, página 192b, Denver, Colorado, USA, 2005. IEEE Press.

[4] E. Alba, F.J. Chicano, C. Cotta, B. Dorronsoro, F. Luna, G. Luque, y A.J. Nebro. *Optimización Inteligente. Técnicas de Inteligencia Computacional para Optimización*, Capítulo 5, Metaheurísticas Secuenciales y Paralelas para Optimización de Problemas Complejos, G. Joya, M. Atencia, A. Ochoa, S. Allende (editores), páginas 185-214. 2004.

[5] E. Alba, J.F. Chicano, B. Dorronsoro, y G. Luque. Diseño de códigos correctores de errores con algoritmos genéticos. En *Actas del Tercer Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB)*, páginas 51–58, Córdoba, España, Febrero de 2004.

[6] E. Alba, K.F. Doerner y B. Dorronsoro, Adapting the Savings based Ant System for non-stationary Vehicle Routing Problems, En *Proceedings of the META 2006*, Tunez, Noviembre de 2006. Edición en CD.

[7] E. Alba y B. Dorronsoro. Auto-adaptación en algoritmos evolutivos celulares. Un nuevo enfoque algorítimico. En *Actas del Segundo Congreso Español sobre Metahurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB)*, páginas 176–185, Gijón, España, Febrero de 2003.

[8] E. Alba y B. Dorronsoro. Solving the vehicle routing problem by using cellular genetic algorithms. En J. Gottlieb and G.R. Raidl, editores, *Evolutionary Computation in Combinatorial Optimization (EvoCOP)*, volumen 3004 de Lecture Notes in Computer Science (LNCS), páginas 11–20, Coimbra, Portugal, Abril de 2004. Springer-Verlag, Heidelberg.

[9] E. Alba y B. Dorronsoro. The exploration/exploitation tradeoff in dynamic cellular genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 9(2):126–142, Abril de 2005.

[10] E. Alba y B. Dorronsoro. Computing nine new best-so-far solutions for capacitated VRP with a cellular genetic algorithm. *Information Processing Letters*, 98(6):225–230, Junio de 2006.

[11] E. Alba y B. Dorronsoro. *Engineering Evolutionary Intelligent Systems*, Capítulo 13, A Hybrid Cellular Genetic Algorithm for the Capacitated Vehicle Routing Problem. Studies in Computational Intelligence. Springer-Verlag, Heidelberg, 2007, por aparecer.

[12] E. Alba, B. Dorronsoro y H. Alfonso. Cellular memetic algorithms. *Journal of Computer Science and Technology*, 5(4):257–263, Diciembre de 2005.

[13] E. Alba, B. Dorronsoro y H. Alfonso. Cellular memetic algorithms evaluated on SAT. En *XI Congreso Argentino de Ciencias de la Computación (CACIC)*, 2005. Edición en CD.

[14] E. Alba, B. Dorronsoro, M. Giacobini y M. Tomassini. *Handbook of Bioinspired Algorithms and Applications*, Capítulo 7, Decentralized Cellular Evolutionary Algorithms, páginas 103–120, 2006. CRC Press.

[15] E. Alba, B. Dorronsoro, F. Luna, A.J. Nebro, P. Bouvry y L. Hogie. A cellular multi-objective genetic algorithm for optimal broadcasting strategy in metropolitan MANETs. *Computer Communications*, por aparecer, 2006.

[16] E. Alba, J. Madera, B. Dorronsoro, A. Ochoa y M. Soto, Theory and Practice of Cellular MUDA for Discrete Optimization, En *Proceedings of the IX Parallel Problem Solving from Nature (PPSN IX)*, T.P. Runarsson et al. (eds.), volumen 4193 de Lecture Notes in Computer Science, páginas 242-251, 2006. Springer-Verlag, Heidelberg. **Premio al mejor póster**.

[17] B. Dorronsoro y E. Alba. A Simple Cellular Genetic Algorithm for Continuous Optimization, En *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, Gary Yen (ed.), páginas 2838-2844, Julio 2006.

[18] B. Dorronsoro, E. Alba, M. Giacobini y M. Tomassini. The influence of grid shape and asynchronicity on cellular evolutionary algorithms. En Y. Shi, editor, *Proc. IEEE International Conference on Evolutionary Computation (CEC)*, páginas 2152–2158, Portland, Oregon, Junio de 2004. IEEE Press.

[19] B. Dorronsoro, A.J. Nebro, D. Arias y E. Alba. Un Algoritmo Genético Híbrido Paralelo para Instancias Complejas del Problema VRP. En *Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB)*, por aparecer, Febrero de 2007.

[20] J.J. Durillo, A.J. Nebro, F. Luna, B. Dorronsoro y E. Alba. jMetal: A Java Framework for Developing Multiobjective Optimization Metaheuristics. Informe Técnico ITI-2006-10, Dpto. de Lenguajes y CC.CC., Universidad de Málaga, 2006.

[21] S. Janson, E. Alba, B. Dorronsoro y M. Middendorf. Hierarchical Cellular Genetic Algorithm, En *Evolutionary Computation in Combinatorial Optimization EvoCOP06*, volumen 3906 de Lecture Notes in Computer Science, J. Gottlieb and G.R. Raidl (eds.), páginas 111-122, Budapest, Hungría, Abril de 2006. Springer-Verlag Heidelberg.

[22] F. Luna, B. Dorronsoro, A.J. Nebro, E. Alba y P. Bouvry. *Handbook on Mobile Ad Hoc and Pervasive Communications*, Capítulo Multiobjective Metaheuristics to Optimize the Broadcasting in MANETs, por aparecer. Diciembre 2006. American Scientific Publishers, USA.

[23] F. Luna, A.J. Nebro, B. Dorronsoro, E. Alba, P. Bouvry y L. Hogie, Optimal Broadcasting in Metropolitan MANETs Using Multiobjective Scatter Search, En *3rd European Workshop on Evolutionary Computation in Communication, Networks and Connected System EvoCOMNET, en EvoWorkshops06*, volumen 3907 de Lecture Notes in Computer Science, Rothlauf and G. D. Smith (eds.), páginas 255-266, Budapest, Hungría, Abril de 2006. Springer-Verlag Heidelberg.

[24] G. Luque, E. Alba y B. Dorronsoro. *Parallel Genetic Algorithms*, Capítulo 5, Parallel Metaheuristics: A New Class of Algorithms, páginas 107–125. John Wiley and Sons, 2005.

[25] J. Madera y B. Dorronsoro. *Metaheuristic Procedures for Training Neural Networks*, Capítulo 5, Estimation of Distribution Algorithms, Operations Research/Computer Science Interfaces Series, páginas 87-108, 2006. Springer-Verlag Heidelberg.

[26] A.J. Nebro, J.J. Durillo, F. Luna, B. Dorronsoro y E. Alba. A Cellular Genetic Algorithm for Multiobjective Optimization, En *Proceedings of the NICSO 2006*, D.A. Pelta and N. Krasnogor (eds.), páginas 25-36, 2006.

[27] A.J. Nebro, J.J. Durillo, F. Luna, B. Dorronsoro y E. Alba. A Study of Strategies for Neigborhood Replacement and Archive Feedback in a Multiobjective Cellular Genetic Algorithm, En *Proceedings of the Evolutionary Multi-criterion Optimization (EMO)*, Lecture Notes in Computer Science, por aparecer, Marzo de 2007, Springer-Verlag Heidelberg.

[28] A.J. Nebro, J.J. Durillo, F. Luna, B. Dorronsoro y E. Alba. MOCell: A Cellular Genetic Algorithm for Multiobjective Optimization. *International Journal of Intelligent Systems*, por aparecer, 2007.

[29] A.J. Nebro, F. Luna, E. Alba, A. Beham y B. Dorronsoro. AbYSS: Adapting scatter search for multiobjective optimization. Informe Técnico ITI-2006-02, Dpto. de Lenguajes y CC.CC., Universidad de Málaga, 2006.

[30] F. Xhafa, E. Alba y B. Dorronsoro. Efficient batch job scheduling in grids using cellular memetic algorithms. En *Nature Inspired Distributed Computing (NIDISC) sessions of the International Parallel and Distributed Processing Simposium (IPDPS) Workshop*. Por aparecer, 2007, IEEE Press.

Bibliografía

[1] http://branchandcut.org/VRP/data/.

[2] Java language specification. http://java.sun.com.

[3] ProActive official web site. http://www-sop.inria.fr/oasis/proactive/.

[4] B. Abdalhaq, A. Cortés, T. Margalef, and E. Luque. Evolutionary optimization techniques on computational grids. In *International Conference on Computational Science*, volume 2393 of *Lecture Notes in Computer Science (LNCS)*, pages 513—-522. Springer-Verlag, Heidelberg, 2002.

[5] D.H. Ackley. *A Connectionist Machine for Genetic Hillclimbing*. Kluwer Academic Publishers, Boston, MA, 1987.

[6] P. Adamidis and V. Petridis. Co-operating populations with different evolution behaviours. In *Proc. IEEE International Conference on Evolutionary Computation (CEC)*, pages 188–191. IEEE Press, 1996.

[7] E. Alba. *Análisis y Diseño de Algoritmos Genéticos Paralelos Distribuidos*. PhD thesis, Universidad de Málaga, Málaga, Febrero 1999.

[8] E. Alba. *Parallel Metaheuristics: A New Class of Algorithms*. Wiley, October 2005.

[9] E. Alba, H. Alfonso, and B. Dorronsoro. Advanced models of cellular genetic algorithms evaluated on SAT. In *Proc. of the Genetic and Evolutionary Computation COnference (GECCO)*, pages 1123–1130, Washington D.C. USA, Junio 25–29 2005. ACM Press.

[10] E. Alba, P. Bouvry, B. Dorronsoro, F. Luna, and A.J. Nebro. A cellular multi-objective genetic algorithm for optimal broadcasting strategy in metropolitan MANETs. In *Nature Inspired Distributed Computing (NIDISC) sessions of the International Parallel and Distributed Processing Simposium (IPDPS) Workshop*, page 192b, Denver, Colorado, USA, 2005.

[11] E. Alba, J.F. Chicano, B. Dorronsoro, and G. Luque. Diseño de códigos correctores de errores con algoritmos genéticos. In *Actas del Tercer Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB)*, pages 51–58, Córdoba, España, 2004. (Spanish).

[12] E. Alba, K.F. Doerner, and B. Dorronsoro. Adapting the savings based ant system for non-stationary vehicle routing problems. In *Proceedings of the META 2006*, Hammamet, Tunez, Noviembre 2006. Edición en CD.

[13] E. Alba and B. Dorronsoro. Auto-adaptación en algoritmos evolutivos celulares. Un nuevo enfoque algorítimico. In *Actas del Segundo Congreso Español sobre Metahurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB)*, pages 176–185, Gijón, España, 2003.

[14] E. Alba and B. Dorronsoro. Solving the vehicle routing problem by using cellular genetic algorithms. In J. Gottlieb and G.R. Raidl, editors, *Evolutionary Computation in Combinatorial Optimization (EvoCOP)*, volume 3004 of *Lecture Notes in Computer Science (LNCS)*, pages 11–20, Coimbra, Portugal, 5-7 April 2004. Springer-Verlag, Heidelberg.

[15] E. Alba and B. Dorronsoro. The exploration/exploitation tradeoff in dynamic cellular evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 9(2):126–142, April 2005.

[16] E. Alba and B. Dorronsoro. Computing nine new best-so-far solutions for capacitated VRP with a cellular genetic algorithm. *Information Processing Letters*, 98(6):225–230, June 2006.

[17] E. Alba and B. Dorronsoro. *Engineering Evolutionary Intelligent Systems*, chapter 13, A Hybrid Cellular Genetic Algorithm for the Capacitated Vehicle Routing Problem. Studies in Computational Intelligence. Springer-Verlag, Heidelberg, 2007. por aparecer.

[18] E. Alba, B. Dorronsoro, and H. Alfonso. Cellular memetic algorithms. *Journal of Computer Science and Technology*, 5(4):257–263, Diciembre 2005.

[19] E. Alba, B. Dorronsoro, and H. Alfonso. Cellular memetic algorithms evaluated on SAT. In *XI Congreso Argentino de Ciencias de la Computación (CACIC)*, 2005. Edición en CD.

[20] E. Alba, B. Dorronsoro, M. Giacobini, and M. Tomassini. *Handbook of Bioinspired Algorithms and Applications*, chapter 7, Decentralized Cellular Evolutionary Algorithms, pages 103–120. CRC Press, 2006.

[21] E. Alba, B. Dorronsoro, F. Luna, A.J. Nebro, P. Bouvry, and L. Hogie. A cellular multi-objective genetic algorithm for optimal broadcasting strategy in metropolitan MANETs. *Computer Communications*, 2006. por aparecer.

[22] E. Alba, M. Giacobini, M. Tomassini, and S. Romero. Comparing synchronous and asynchronous cellular genetic algorithms. In J.J. Merelo et al., editor, *Proc. of the International Conference on Parallel Problem Solving from Nature VII (PPSN-VII)*, volume 2439 of *Lecture Notes in Computer Science (LNCS)*, pages 601–610, Granada, Spain, 2002. Springer-Verlag, Heidelberg.

[23] E. Alba, F. Luna, and A.J. Nebro. Advances in parallel heterogeneous genetic algorithms for continuous optimization. *International Journal of Applied Mathematics and Computer Science*, 14(3):101–117, 2004.

[24] E. Alba and G. Luque. Growth curves and takeover time in distributed evolutionary algorithms. In K. Deb et al., editor, *Proc. of the Genetic and Evolutionary Computation COnference (GECCO)*, volume 3102 of *Lecture Notes in Computer Science (LNCS)*, pages 864–876. Springer-Verlag, Heidelberg, 2004.

[25] E. Alba, J. Madera, B. Dorronsoro, A. Ochoa, and M. Soto. Theory and practice of cellular UMDA for discrete optimization. In T.P. Runarsson et al., editor, *Proc. of the International Conference on Parallel Problem Solving from Nature IX (PPSN-IX)*, volume 4193 of *Lecture Notes in Computer Science (LNCS)*, pages 242–251, Reykjavik, Iceland, September 2006. Springer-Verlag, Heidelberg.

[26] E. Alba and J.F. Saucedo. Panmictic versus decentralized genetic algorithms for non-stationary problems. In *Sixth Metaheuristics International Conference (MIC)*, pages 7–12, Viena, Austria, 2005.

[27] E. Alba and M. Tomassini. Parallelism and evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 6(5):443–462, October 2002.

[28] E. Alba and J.M. Troya. Cellular evolutionary algorithms: Evaluating the influence of ratio. In M. Schoenauer, editor, *Proc. of the International Conference on Parallel Problem Solving from Nature VI (PPSN-VI)*, volume 1917 of *Lecture Notes in Computer Science (LNCS)*, pages 29–38. Springer-Verlag, Heidelberg, 2000.

[29] E. Alba and J.M. Troya. Improving flexibility and efficiency by adding parallelism to genetic algorithms. *Statistics and Computing*, 12(2):91–114, 2002.

[30] P.J. Angeline. Adaptive and self-adaptive evolutionary computations. In Marimuthu Palaniswami and Yianni Attikiouzel, editors, *Computational Intelligence: A Dynamic Systems Perspective*, pages 152–163. IEEE Press, 1995.

[31] J. Arabas, Z. Michalewicz, and J. Mulawka. GAVaPS - a genetic algorithm with varying population size. In *Proc. IEEE International Conference on Evolutionary Computation (CEC)*, volume 1, pages 73–78, 1994.

[32] P. Augerat, J.M. Belenguer, E. Benavent, A. Corberán, D. Naddef, and G. Rinaldi. Computational results with a branch and cut code for the capacitated vehicle routing problem. Research Report 949-M, Universite Joseph Fourier, Grenoble, France, 1995.

[33] S. Baluja. Structure and performance of fine-grain parallelism in genetic search. In S. Forrest, editor, *Proc. of the Fifth International Conference on Genetic Algorithms (ICGA)*, pages 155–162. Morgan Kaufmann, 1993.

[34] S. Baluja. Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Technical Report CMU-CS-94-163, Carnegie Mellon University, 1994.

[35] W. Banzhaf. The "molecular" traveling salesman. *Biological Cybernetics*, 64:7–14, 1990.

[36] T. Bäck. Self-adaptation in genetic algorithms. In F.J. Varela and P. Bourgine, editors, *Proc. of the 1st European Conference on Artificial Life*, pages 263–271, Cambridge, MA, 1992. The MIT Press.

[37] T. Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms.* Oxford University Press, New York, 1996.

[38] T. Bäck. Introduction to the special issue: Self-adaptation. *Evolutionary Computation*, 9(2):iii–iv, 2001.

[39] T. Bäck, A.E. Eiben, and M.E. Vink. A superior evolutionary algorithm for 3-SAT. In V.W. Porto, N. Saravanan, D. Waagen, and A.E. Eiben, editors, *International Conference on Evolutionary Programming VII*, volume 1477 of *Lecture Notes in Computer Science (LNCS)*. Springer-Verlag, Heidelberg, 1998.

[40] T. Bäck, D.B. Fogel, and Z. Michalewicz, editors. *Handbook of Evolutionary Computation*. Oxford University Press, 1997.

[41] T. Bäck, G. Rudolf, and H.-P. Schwefel. Evolutionary programming and evolution strategies: similarities and differences. In D.B. Fogel and W. Atmar, editors, *Proc. of the Second Conference on Evolutionary Programming*, pages 11–22, La Jolla, California, 1993. Evolutionary Programming Society.

[42] J. Berger and M. Barkaoui. A hybrid genetic algorithm for the capacitated vehicle routing problem. In E. Cantú-Paz, editor, *Proc. of the Genetic and Evolutionary Computation COnference (GECCO)*, volume 2723 of *Lecture Notes in Computer Science (LNCS)*, pages 646–656, Illinois, Chicago, USA, 2003. Springer-Verlag, Heidelberg.

[43] F. Berman, G. Fox, and A. Hey. *Grid Computing. Making the Global Infrastructure a Reality*. Communications Networking and Distributed Systems. John Wiley & Sons, 2003.

[44] A. Bethke. Comparison of genetic algorithms and gradient based optimizers on parallel processors: Efficiency of use of precessing capacity. Technical Report 197, Ann Arbor, University of Michigan, 1976.

[45] S. Bleuler, M. Laumanns, L. Thiele, and E. Zitzler. PISA – a platform and programming language independent interface for search algorithms. In C.M. Fonseca, P.J. Fleming, E. Zitzler, K. Deb, and L. Thiele, editors, *Proc. of the International Conference on Evolutionary Multi-criterion Optimization (EMO)*, volume 2632 of *Lecture Notes in Computer Science (LNCS)*, pages 494–508, Faro, Portugal, 2003. Springer-Verlag, Heidelberg.

[46] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.

[47] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence. From Natural to Artificial Systems*. Oxford University Press, 1999.

[48] P.A.N. Bosman and D. Thierens. Advancing continuous IDEAs with mixture distributions and factorization selection metrics. In *Optimization by Building and Using Probabilistics Models, OBUPM Workshop at the Genetic and Evolutionary Computation Conference GECCO*, pages 208–212, 2001.

[49] H.J. Bremermann. *Self-Organizing Systems*, chapter Optimization Trough Evolution and Resombination, pages 93–106. Spartan Books, Washington DC, 1962.

[50] S. Cahon, N. Melab, and E-G. Talbi. ParadisEO: A framework for the reusable design of parallel and distributed metaheuristics. *Journal of Heuristics*, 10(3):357—-380, May 2004.

[51] E. Cantú-Paz. *Efficient and Accurate Parallel Genetic Algorithms*, volume 1 of *Book Series on Genetic Algorithms and Evolutionary Computation*. Kluwer Academic Publishers, 2nd edition, 2000.

[52] E. Cantú-Paz. Feature subset selection by estimation of distribution algorithms. In W.B. Langdon et al., editor, *Proc. of the Genetic and Evolutionary Computation COnference (GECCO)*, pages 303–310, San Francisco, CA, 2002. Morgan Kaufmann.

[53] M. Capcarrère, M. Tomassini, A. Tettamanzi, and M. Sipper. A statistical study of a class of cellular evolutionary algorithms. *Evolutionary Computation*, 7(3):255–274, 1999.

[54] U.K. Chakraborty, K. Deb, and M. Chakraborty. Analysis of selection algorithms: A markov chain approach. *Evolutionary Computation*, 4:133–167, 1997.

[55] K. Chellapilla. Combining mutation operators in evolutionary programming. *IEEE Transactions on Evolutionary Computation*, 2(3):91–96, September 1998.

[56] H. Chen, N.S. Flann, and D.W. Watson. Parallel genetic simulated annealing: A massively parallel SIMD algorithm. *IEEE Transactions on Parallel and Distributed Systems*, 9(2):126–136, 1998.

[57] N. Christofides and S. Eilon. An algorithm for the vehicle dispatching problem. *Operations Research Quartely*, 20:309—-318, 1969.

[58] N. Christofides, A. Mingozzi, and P. Toth. *Combinatorial Optimization*, chapter The Vehicle Routing Problem, pages 315–338. John Wiley & Sons, 1979.

[59] C.A. Coello and G. Toscano. Multiobjective optimization using a micro-genetic algorithm. In *Proc. of the Genetic and Evolutionary Computation COnference (GECCO)*, pages 274–282, 2001.

[60] C.A. Coello, D.A. Van Veldhuizen, and G.B. Lamont. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Genetic Algorithms and Evolutionary Computation. Kluwer Academic Publishers, 2002.

[61] R.J. Collins and D.R. Jefferson. Selection in massively parallel genetic algorithms. In R.K. Belew and L.B. Booker, editors, *Proc. of the Fourth International Conference on Genetic Algorithms (ICGA)*, pages 249–256, San Diego, CA, USA, 1991. Morgan Kaufmann.

[62] S.A. Cook. The complexity of theorem-proving procedures. *Proc. of the Third Anual ACM Symp. on the Theory of Computing*, pages 151–158, 1971.

[63] J.-F. Cordeau, M. Gendreau, A. Hertz, G. Laporte, and J.-S. Sormany. *Logistics Systems: Design and Optimization*, chapter 9. New Heuristics for the Vehicle Routing Problem, pages 279–298. Springer-Verlag, Heidelberg, 2004.

[64] C. Cotta, E. Alba, and J.M. Troya. Un estudio de la robustez de los algoritmos genéticos paralelos. *Revista Iberoamericana de Inteligencia Artificial*, 98(5):6–13, 1998.

[65] N.L. Cramer. A representation for the adaptive generation of simple sequential programs. In J.J. Grefenstette, editor, *Proc. of the First International Conference on Genetic Algorithms and their Applications*, pages 183–187, Carnegie-Mellon University, Pittsburgh, PA, USA, July 24–26 1985.

[66] G.A. Croes. A method for solving traveling salesman problems. *Operations Research*, 6:791–812, 1958.

[67] J.M. Daida, S.J. Ross, and B.C. Hannan. Biological symbiosis as metaphor for computational hybridization. In L.J. Eshelman, editor, *Proc. of the Sixth International Conference on Genetic Algorithms (ICGA)*, pages 328–335. Morgan Kaufmann, 1995.

[68] G.B. Dantzing and R.H. Ramster. The truck dispatching problem. *Management Science*, 6:80–91, 1959.

[69] C. Darwin. *On the Origin of Species by Means of Natural Selection.* John Murray, Londres, 1859.

[70] Y. Davidor. A naturally occurring niche and species phenomenon: The model and first results. In R.K. Belew and L.B. Booker, editors, *Proc. of the Fourth International Conference on Genetic Algorithms (ICGA)*, pages 257–263, San Diego, CA, July 1991. Morgan Kaufmann.

[71] Y. Davidor, T. Yamada, and R. Nakano. The ECOlogical framework II: Improving GA performance at virtually zero cost. In S. Forrest, editor, *Proc. of the Fifth International Conference on Genetic Algorithms (ICGA)*, pages 171–176. Morgan Kaufmann, 1993.

[72] L. Davis. Adapting operator probabilities in genetic algorithms. In J.D. Schaffer, editor, *Proc. of the Third International Conference on Genetic Algorithms (ICGA)*, pages 61–69. Morgan Kaufmann, 1989.

[73] J.S. De Bonet, C.L. Isbell, and P. Viola. MIMIC: Finding optima by estimating probability densities. In M. Jordan, M. Mozer, and M. Perrone, editors, *Advances in Neural Information Processing Systems*, volume 9, pages 424–430, Denver, USA, 1997. The MIT Press.

[74] K. De Jong and J. Sarma. On decentralizing selection algorithms. In L. Eshelman, editor, *Proc. of the Sixth International Conference on Genetic Algorithms (ICGA)*, pages 17–23, San Francisco, CA, 1995. Morgan Kaufmann.

[75] K.A. De Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems.* PhD thesis, University of Michigan, 1975. Ann Arbor.

[76] K.A. De Jong. *Evolutionary Computation. A Unified Approach.* The MIT Press, 2006.

[77] K.A. De Jong and W.M. Spears. Using genetic algorithm to solve NP-complete problems. In J.D. Schaffer, editor, *Proc. of the Third International Conference on Genetic Algorithms (ICGA)*, pages 124–132. Morgan Kaufmann, 1989.

[78] K. Deb. *Multi-Objective Optimization using Evolutionary Algorithms.* Wiley, 2001.

[79] K. Deb and R.B. Agrawal. Simulated binary crossover for continuous search space. *Complex Systems*, 9:115–148, 1995.

[80] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[81] G. Dick. A comparison of localised and global niching methods. In *Annual Colloquium of the Spatial Information Research Centre (SIRC)*, pages 91–101, Dunedin, New Zealand, Novenber 2005.

[82] G. Dick and P. Whigham. The behaviour of genetic drift in a spatially-structured evolutionary algorithm. In *Proc. IEEE International Conference on Evolutionary Computation (CEC)*, volume 2, pages 1855–1860. IEEE Press, 2005.

[83] M. Dorigo and T. Stützle. *Ant Colony Optimization.* The MIT Press, 2004.

[84] B. Dorronsoro. http://neo.lcc.uma.es/radi-aeb/WebVRP/.

[85] B. Dorronsoro. http://neo.lcc.uma.es/cEA-web/.

[86] B. Dorronsoro and E. Alba. A simple cellular genetic algorithm for continuous optimization. In G. Yen, editor, *Proc. of the IEEE Conference on Evolutionary Computation (CEC), IEEE World Congress on Computational Intelligence (WCCI)*, pages 2838–2844, Vancouver, Canada, July 2006. IEEE Press.

[87] B. Dorronsoro, E. Alba, M. Giacobini, and M. Tomassini. The influence of grid shape and asynchronicity on cellular evolutionary algorithms. In Y. Shi, editor, *Proc. IEEE International Conference on Evolutionary Computation (CEC)*, pages 2152–2158, Portland, Oregon, June 20–23 2004. IEEE Press.

[88] B. Dorronsoro, A.J. Nebro, D. Arias, and E. Alba. Un algoritmo genético híbrido paralelo para instancias complejas del problema VRP. In *Actas del Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB)*, 2007. por aparecer.

[89] S. Droste, T. Jansen, and I. Wegener. A natural and simple function which is hard for all evolutionary algorithms. In *Proc. of the Asia-Pacific Conference on Simulated Evolution and Learning (SEAL)*, pages 2704–2709, Nagoya, Japan, 2000.

[90] T. Duncan. Experiments in the use of neighbourhood search techniques for vehicle routing. Technical Report AIAI-TR-176, Artificial Intelligence Applications Institute, University of Edinburgh, Edinburgh, 1995.

[91] J.J. Durillo. http://neo.lcc.uma.es/software/metal/.

[92] J.J. Durillo, A.J. Nebro, F. Luna, B. Dorronsoro, and E. Alba. jMetal: A java framework for developing multiobjective optimization metaheuristics. Technical Report ITI-2006-10, Dpto. de Lenguajes y CC.CC., Universidad de Málaga, 2006.

[93] E. Alba and the MALLBA Group. MALLBA: A library of skeletons for combinatorial optimization. In R.F.B. Monien, editor, *Proc. of the Euro-Par*, volume 2400 of *Lecture Notes in Computer Science (LNCS)*, pages 927–932, Paderborn, Germany, 2002. Springer-Verlag, Heidelberg.

[94] B. Eckel. *Thinking in Java.* MindView, 2002.

[95] A.E. Eiben and J.K. Van der Hauw. Solving 3-SAT with adaptive genetic algorithms. In *Proc. of the IEEE Conference on Evolutionary Computation (CEC), IEEE World Congress on Computational Intelligence (WCCI)*, pages 81–86. IEEE Press, 1997.

[96] S.E. Eklund. Empirical studies of neighborhood shapes in the massively parallel diffusion model. In G. Bittencourt and G. Ramalho, editors, *Brazilian Symposium on Artificial Intelligence (SBIA)*, volume 2507 of *Lecture Notes in Artificial Intelligence (LNAI)*, pages 185–194. Springer-Verlag, Heidelberg, 2002.

[97] L.J. Eshelman, K.E. Mathias, and J.D. Schaffer. Convergence controlled variation. In R. Belew and M. Vose, editors, *Foundations of Genetic Algorithms IV (FOGA)*, pages 203–224, San Mateo, CA, 1989. Morgan Kaufmann.

[98] L.J. Eshelman and J.D. Schaffer. Real coded genetic algorithms and interval schemata. In L.D. Whitley, editor, *Foundations of Genetic Algorithms II (FOGA)*, pages 187—-202, San Mateo, 1993. Morgan Kaufmann.

[99] M.L. Fisher. Optimal solution of vehicle routing problems using minimum k-trees. *Operations Research*, 42—44:626–642, 1994.

[100] M.J. Flynn. Very high speed computing systems. *Proc. IEEE*, 54:1901–1909, 1966.

[101] D.B. Fogel. An evolutionary approach to the traveling salesman problem. *Biological Cybernetics*, 60:139–144, 1988.

[102] L.J. Fogel. Autonomous automata. *Industrial Research*, 4:14–19, 1962.

[103] G. Folino, C. Pizzuti, and G. Spezzano. Combining cellular genetic algorithms and local search for solving satisfiability problems. In *Proc. of the IEEE International Conference on Tools with Artificial Intelligence*, pages 192–198, Taipei, Taiwan, 1998. IEEE Press.

[104] G. Folino, C. Pizzuti, and G. Spezzano. Parallel hybrid method for SAT that couples genetic algorithms and local search. *IEEE Transactions on Evolutionary Computation*, 5(4):323–334, August 2001.

[105] G. Folino, C. Pizzuti, and G. Spezzano. A scalable cellular implementation of parallel genetic programming. *IEEE Transactions on Evolutionary Computation*, 7(1):37–53, February 2003.

[106] G. Folino and G. Spezzano. A cellular environment for steering high performance scientific computations. In E.H. D'Hollander, J.R. Joubert, F.J. Peters, and H. Sips, editors, *Proc. of the International Conference on Parallel Computing: Fundamentals & Applications (ParCo)*, pages 493–500. Imperial College Press, April 2000.

[107] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, San Fransisco, 1999.

[108] A.S. Fraser. Simulation of genetic systems by automatic digital computers II: Effects of linkage on rates under selection. *Australian Journal of Biological Sciences*, 10:492–499, 1957.

[109] R. Fukasawa, J. Lysgaard, M.P. de Aragão, M. Reis, E. Uchoa, and R.F. Werneck. Robust branch-and-cut-and-price for the capacitated vehicle routing problem. In *Integer Programming and Combinatorial Optimization (IPCO)*, volume 3064 of *Lecture Notes in Computer Science (LNCS)*, pages 1–15, New Yorj, USA, July 2004. Springer-Verlag, Heidelberg.

[110] L.M. Gambardella, E. Taillard, and G. Agazzi. *New Ideas in Optimization*, chapter MACS-VRPTW: A Multiple Ant Colony System for Vehicle Routing Problems with Time Windows, pages 63–76. McGraw-Hill, 1999.

[111] M. Garey and D. Johnson. *Computers and Intractability: a Guide to the Theory of NP-completeness*. Freeman, San Francisco, CA, 1979.

[112] H.G. Gauch Jr. *Scientific Method in Practice*. Cambridge, 1st edition, 2002.

[113] M. Gendreau, A. Hertz, and G. Laporte. A tabu search heuristic for the vehicle routing problem. Technical Report CRT-777, Centre de Recherche sur les Transports - Université de Montréal, Montréal, Canada, 1991.

[114] M. Gendreau, A. Hertz, and G. Laporte. A tabu search heuristic for the vehicle routing problem. *Management Science*, 40:1276–1290, 1994.

[115] M. Giacobini, E. Alba, A. Tettamanzi, and M. Tomassini. Modelling selection intensity for toroidal cellular evolutionary algorithms. In *Proc. of the Genetic and Evolutionary Computation COnference (GECCO)*, volume 3102 of *Lecture Notes in Computer Science (LNCS)*, pages 1138–1149. Springer-Verlag, Heidelberg, 2004.

[116] M. Giacobini, E. Alba, and M. Tomassini. Selection intensity in asynchronous cellular evolutionary algorithms. In *Proc. of the Genetic and Evolutionary Computation COnference (GECCO)*, pages 955–966. Springer-Verlag, Heidelberg, 2003.

[117] M. Giacobini, M. Tomassini, and A. Tettamanzi. Modelling selection intensity for linear cellular evolutionary algorithms. In P. Liardet et al., editor, *Proc. of the International Conference on Artificial Evolution*, volume 2936 of *Lecture Notes in Computer Science (LNCS)*, pages 345–356. Springer-Verlag, Heidelberg, 2003.

[118] M. Giacobini, M. Tomassini, and A. Tettamanzi. Takeover time curves in random and small-world structured populations. In *Proc. of the Genetic and Evolutionary Computation COnference (GECCO)*, pages 1333–1340, Washington D.C. USA, June 25–29 2005. ACM Press.

[119] M. Giacobini, M. Tomassini, A.G.B. Tettamanzi, and E. Alba. Selection intensity in cellular evolutionary algorithms for regular lattices. *IEEE Transactions on Evolutionary Computation*, 9(5):489–505, October 2005.

[120] F. Glover and M. Laguna. *Tabu Search.* Kluwer Academic Publishers, Boston, 1997.

[121] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Publishing Company, 1989.

[122] D.E. Goldberg and K. Deb. A comparative analysis of selection schemes used in genetic algorithms. In G.J.E. Rawlins, editor, *Foundations of Genetic Algorithms I (FOGA)*, pages 69–93, San Mateo, CA, USA, 1991. Morgan Kaufmann.

[123] D.E. Goldberg, K. Deb, and J. Horn. Massively multimodality, deception and genetic algorithms. In R. Männer and B. Manderick, editors, *Proc. of the International Conference on Parallel Problem Solving from Nature II (PPSN-II)*, pages 37–46. North-Holland, 1992.

[124] D.E. Goldberg, K. Deb, H. Kargupta, and G. Harik. Rapid, accurate optimization of difficult problems using fast messy genetic algorithms. In S. Forrest, editor, *Proc. of the Fifth International Conference on Genetic Algorithms (ICGA)*, pages 56–64. Morgan Kaufmann, 1993. San Mateo.

[125] B.L. Golden, E.A. Wasil, J.P. Kelly, and I-M. Chao. *Fleet Management and Logistics*, chapter The Impact of Metaheuristics on Solving the Vehicle Routing Problem: algorithms, problem sets, and computational results, pages 33–56. Kluwer Academic Publishers, Boston, 1998.

[126] V. Gordon, K. Mathias, and D. Whitley. Cellular genetic algorithms as function optimizers: Locality effects. In *ACM Symposium on Applied Computing (SAC)*, pages 237–241. ACM Press, 1994.

[127] V. Gordon, R. Pirie, A. Wachter, and S. Sharp. Terrain-based genetic algorithm (tbga): Modeling parameter space as terrain. In *Proc. of the Genetic and Evolutionary Computation COnference (GECCO)*, pages 229–235, 1999.

[128] V. Gordon, D. Whitley, and A. Böhm. Dataflow parallelism in genetic algorithms. In R. Männer and B. Manderick, editors, *Proc. of the International Conference on Parallel Problem Solving from Nature II (PPSN-II)*, pages 533–542. North-Holland, 1992.

[129] V.S. Gordon and J. Thein. Visualization tool for a terrain-based genetic algorithm. In *IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 401–406. IEEE Press, 2004.

[130] V.S. Gordon and D. Whitley. Serial and parallel genetic algorithms as function optimizers. In S. Forrest, editor, *Proc. of the Fifth International Conference on Genetic Algorithms (ICGA)*, pages 177–183. Morgan Kaufmann, 1993.

[131] M. Gorges-Schleuter. ASPARAGOS - an asynchronous parallel genetic optimization strategy. In J.D. Schaffer, editor, *Proc. of the Third International Conference on Genetic Algorithms (ICGA)*, pages 422–428. Morgan Kaufmann, 1989.

[132] M. Gorges-Schleuter. Comparison of local mating strategies in massively parallel genetic algorithms. In R. Männer and B. Manderick, editors, *Proc. of the International Conference on Parallel Problem Solving from Nature II (PPSN-II)*, pages 553–562. North-Holland, 1992.

[133] M. Gorges-Schleuter. Asparagos96 and the traveling salesman problem. In *Proc. IEEE International Conference on Evolutionary Computation (CEC)*, pages 171–174. IEEE Press, 1997.

[134] M. Gorges-Schleuter. A comparative study of global and local selection in evolution strategies. In *Proc. of the International Conference on Parallel Problem Solving from Nature V (PPSN-V)*, volume 1498 of *Lecture Notes in Computer Science (LNCS)*, pages 367–377. Springer-Verlag, Heidelberg, 1998.

[135] M. Gorges-Schleuter. An analysis of local selection in evolution strategies. In *Proc. of the Genetic and Evolutionary Computation COnference (GECCO)*, volume 1, pages 847–854, San Francisco, CA, USA, 1999. Morgan Kaufmann.

[136] J. Gottlieb, E. Marchiori, and C. Rossi. Evolutionary algorithms for the satisfiability problem. *Evolutionary Computation*, 10(2):35–502, 2002.

[137] J. Gottlieb and N. Voss. Representations, fitness functions and genetic operators for the satisfiability problem. In *Artificial Evolution*, Lecture Notes in Computer Science (LNCS), pages 55–68. Springer-Verlag, Heidelberg, 1998.

[138] R.L. Graham. Bounds on multiprocessor timing anomalies. *SIAM Journal of Applied Mathematics*, 17:416–429, 1969.

[139] A.O. Griewangk. Generalized descent of global optimization. *Journal of Optimization, Theory, and Applications*, 34:11–39, 1981.

[140] C. Grimme and K. Schmitt. Inside a predator-prey model for multi-objective optimization: A second study. In M. Cattolico, editor, *Proc. of the Genetic and Evolutionary Computation COnference (GECCO)*, pages 707–714, Seattle, Washington, USA, July 8–12 2006. ACM Press.

[141] F. Herrera, E. Herrera-Viedma, M. Lozano, and J.L. Verdegay. Fuzzy tools to improve genetic algorithms. In *Proc. Second European Congress on Intelligent Techniques and Soft Computing*, pages 1532—-1539, 1994.

[142] F. Herrera and M. Lozano. Gradual distributed real-coded genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 4(1):43–62, April 2000.

[143] F. Herrera, M. Lozano, and J.L. Verdegay. Tackling real-coded genetic algorithms: Operators and tools for the behavioral analysis. *Artificial Intelligence Reviews*, 12(4):265–319, 1998.

[144] D. Hillis. Co-evolving parasites improve simulated evolution as an optimizing procedure. *Physica D*, 42:228–234, 1990.

[145] R. Hinterding, Z. Michalewicz, and A. Eiben. Adaptation in evolutionary computation: A survey. In *Proc. of the IEEE Conference on Evolutionary Computation (CEC), IEEE World Congress on Computational Intelligence (WCCI)*, pages 65–69, 1997.

[146] F. Hoffmeister. *Applied Parallel and Distributed Optimization*, chapter Scalable Parallelism by Evolutionary Algorithms, pages 175–198. Springer-Verlag, Heidelberg, 1991.

[147] L. Hogie, F. Guinand, and P. Bouvry. *The Madhoc Metropolitan Adhoc Network Simulator*. Université du Luxembourg and Université du Havre, France. Available at http://www-lih.univ-lehavre.fr/∼hogie/madhoc/.

[148] L. Hogie, M. Seredynski, F. Guinand, and P. Bouvry. A bandwidth-efficient broadcasting protocol for mobile multi-hop ad hoc networks. In *International Conference on Networking (ICN)*, page 71. IEEE Press, 2006.

[149] J.H. Holland. Outline for a logical theory of adaptive systems. *Journal of the ACM*, 9(3):297–314, 1962.

[150] J.H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, 1975.

[151] S. Huband, L. Barone, R.L. While, and P. Hingston. A scalable multi-objective test problem toolkit. In C.A. Coello, A. Hernández, and E. Zitler, editors, *Proc. of the International Conference on Evolutionary Multi-criterion Optimization (EMO)*, volume 3410 of *Lecture Notes in Computer Science (LNCS)*, pages 280–295, 2005.

[152] T.S. Hussain. An introduction to evolutionary computation. tutorial presentation. CITO Researcher Retreat, May 12-14, Hamilton, Ontario 1998.

[153] H. Ishibuchi, T. Doi, and Y. Nojima. Effects of using two neighborhood structures in cellular genetic algorithms for function optimization. In T.P. Runarsson et al., editor, *Proc. of the International Conference on Parallel Problem Solving from Nature IX (PPSN-IX)*, volume 4193 of *Lecture Notes in Computer Science (LNCS)*, pages 949–958, Reykjavik, Iceland, September 2006. Springer-Verlag, Heidelberg.

[154] S. Janson, E. Alba, B. Dorronsoro, and M. Middendorf. Hierarchical cellular genetic algorithm. In J. Gottlieb and G.R. Raidl, editors, *Evolutionary Computation in Combinatorial Optimization (EvoCOP)*, volume 3906 of *Lecture Notes in Computer Science (LNCS)*, pages 111–122, Budapest, Hungría, April 2006. Springer-Verlag, Heidelberg.

[155] S. Janson and M. Middendorf. A hierarchical particle swarm optimizer and its adaptive variant. *IEEE Systems, Man and Cybernetics - Part B*, 35(6):1272–1282, 2005.

[156] K.A. De Jong, M.A. Potter, and W.M. Spears. Using problem generators to explore the effects of epistasis. In T. Bäck, editor, *Proc. of the Seventh International Conference on Genetic Algorithms (ICGA)*, pages 338–345. Morgan Kaufmann, 1997.

[157] K.A. De Jong, M.A. Potter, and W.M. Spears. Using problem generators to explore the effects of epistasis. In T. Bäck, editor, *Proc. of the Seventh International Conference on Genetic Algorithms (ICGA)*, pages 338–345. Morgan Kaufmann, 1997.

[158] H. Juille and J.B. Pollack. *Advances in Genetic Programming 2*, chapter Massively parallel genetic programming, pages 339–358. The MIT Press, MA, USA, 1996.

[159] R.M. Karp. Probabilistic analysis of partitioning algorithms for the traveling salesman problem in the plane. *Mathematics of Operations Research*, 2:209—-224, 1977.

[160] H.A. Kautz and B. Selman. Planning as satisfiability. In *European Conference on Artificial Intelligence*, pages 359–363, 1992.

[161] S. Khuri, T. Bäck, and J. Heitkötter. An evolutionary approach to combinatorial optimization problems. In *Proc. of the ACM Computer Science Conference*, pages 66–73, Phoenix, Arizona, 1994. ACM Press.

[162] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 4598:671–680, 1983.

[163] M. Kirley. MEA: A metapopulation evolutionary algorithm for multi-objective optimisation problems. In *Proc. IEEE International Conference on Evolutionary Computation (CEC)*, pages 949–956. IEEE Press, 2001.

[164] M. Kirley. A cellular genetic algorithm with disturbances: Optimisation using dynamic spatial interactions. *Journal of Heuristics*, 8:321–342, 2002.

[165] M. Kirley and D.G. Green. An empirical investigation of optimization in dynamic environments using the cellular genetic algorithm. In D. Whitley et al., editor, *Proc. of the Genetic and Evolutionary Computation COnference (GECCO)*, pages 11–18, San Mateo, CA, 2000. Morgan Kaufmann.

[166] M. Kirley, X. Li, and D.G. Green. Investigation of a cellular genetic algorithm that mimics landscape ecology. In X. Yao, editor, *Proc. of the Asia-Pacific Conference on Simulated Evolution and Learning (SEAL)*, volume 1585 of *Lecture Notes in Computer Science (LNCS)*, pages 90–97. Springer-Verlag, Heidelberg, 1999.

[167] J. Knowles and D. Corne. The pareto archived evolution strategy: A new baseline algorithm for multiobjective optimization. In *Proc. IEEE International Conference on Evolutionary Computation (CEC)*, pages 98–105, Piscataway, NJ, 1999. IEEE Press.

[168] J. Knowles and D. Corne. Approximating the nondominated front using the Pareto archived evolution strategy. *Evolutionary Computation*, 8(2):149–172, 2001.

[169] U. Kohlmorgen, H. Schmeck, and K. Haase. Experiences with fine-grained parallel genetic algorithms. *Annals of Operations Research*, 90:302–219, 1999.

[170] T. Krink and R. Thomsen. Self-organized criticality and mass extinction in evolutionary algorithms. In *Proc. IEEE International Conference on Evolutionary Computation (CEC)*, pages 1155–1161, Seoul, Korea, 2001. IEEE Press.

[171] K.W.C. Ku. Enhance the baldwin effect by strengthening the correlation between genetic operators and learning methods. In G. Yen, editor, *Proc. of the IEEE Conference on Evolutionary Computation (CEC), IEEE World Congress on Computational Intelligence (WCCI)*, pages 11071–11077, Vancouver, BC, Canada, July 16-21 2006. IEEE Press.

[172] K.W.C. Ku, M.W. Mak, and W.C. Siu. Adding learning to cellular genetic algorithms for training recurrent neural networks. *IEEE Transactions on Neural Networks*, 10(2):239–252, March 1999.

[173] M. Laguna and R. Martí. *Scatter Search. Methodology and Implementations.* Operations Research/Computer Science Interfaces Series. Kluwer Academic Publishers, Boston, 2003.

[174] P. Larrañaga, R. Etxeberria, J.A. Lozano, and J.M. Peña. Optimization by learning and simulation of Bayesian and Gaussian networks. Technical Report KZZA-IK-4-99, Department of Computer Science and Artificial Intelligence, University of the Basque Country, 1999.

[175] P. Larrañaga, R. Etxeberria, J.A. Lozano, and J.M. Peña. Optimization in continuous domains by learning and simulation of Gaussian networks. In *Proc. of the Genetic and Evolutionary Computation COnference (GECCO)*, pages 201–204, 2000.

[176] P. Larrañaga and J.A Lozano. *Estimation of distribution algorithms. A new tool for evolutionary computation.* Kluwer Academic Publishers, 2001.

[177] P. Larrañaga and J.A. Lozano, editors. *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation.* Kluwer Academic Publishers, 2002.

[178] M. Laumanns, G. Rudolph, and H.P. Schwefel. A spatial predator-prey approach to multiobjective optimization: A preliminary study. In A.E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, editors, *Proc. of the International Conference on Parallel Problem Solving from Nature V (PPSN-V)*, volume 1498 of *Lecture Notes in Computer Science (LNCS)*, page 241–249, 1998.

[179] C.-H. Lee, S.-H. Park, and J.-H. Kim. Topology and migration policy of fine-grained parallel evolutionary algorithms for numerical optimization. In *Proc. IEEE International Conference on Evolutionary Computation (CEC)*, volume 1, pages 70–76. IEEE Press, 2000.

[180] J.K. Lenstra and A.H.G. Rinnooy Kan. Complexity of vehicle routing and scheduling problems. *Networks*, 11:221–227, 1981.

[181] F. Li, B. Golden, and E. Wasil. Very large-scale vehicle routing: new test problems, algorithms, and results. *Computers & Operations Research*, 32:1165–1179, 2005.

[182] X. Li. A real-coded predator-prey genetic algorithm for multiobjective optimization. In C.M. Fonseca et al., editor, *Proc. of the International Conference on Evolutionary Multi-criterion Optimization (EMO)*, volume 2632 of *Lecture Notes in Computer Science (LNCS)*, pages 207–221. Springer-Verlag, Heidelberg, 2003.

[183] X. Li and S. Sutherland. A cellular genetic algorithm simulating predator-prey interactions. In *Proc. of the Third International Conference on Genetic Algorithms (ICGA)*, pages 416–421. Morgan Kaufmann, 2002.

[184] H. Lim and C. Kim. Multicast tree construction and flooding in wireless ad hoc networks. In *ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWIM)*, pages 61–68. ACM Press, 2000.

[185] X. Llorà and J.M. Garrell. Knowledge-independent data mining with fine-grained parallel evolutionary algorithms. In L. Spector et al., editor, *Proc. of the Genetic and Evolutionary Computation COnference (GECCO)*, pages 461–468. Morgan Kaufmann, 2001.

[186] H.R. Lourenco, O. Martin, and T. Stützle. *Handbook of Metaheuristics*, chapter Iterated Local Search, pages 321–353. Kluwer Academic Publishers, Norwell, MA, 2002.

[187] F. Luna, B. Dorronsoro, A.J. Nebro, E. Alba, and P. Bouvry. *Handbook on Mobile Ad Hoc and Pervasive Communications*, chapter Multiobjective Metaheuristics to Optimize the Broadcasting in MANETs. American Scientific Publishers, USA, Diciembre 2006. por aparecer.

[188] F. Luna, A.J. Nebro, B. Dorronsoro, E. Alba, P. Bouvry, and L. Hogie. Optimal broadcasting in metropolitan MANETs using multiobjective scatter search. In *European Workshop on Evolutionary Computation in Communication, Networks and Connected System (EvoCOMNET), en EvoWorkshops*, volume 3907 of *Lecture Notes in Computer Science (LNCS)*, pages 255–266, Budapest, Hungría, April 2006. Springer-Verlag, Heidelberg.

[189] Z. Luo and H. Liu. Cellular genetic algorithms and local search for 3-SAT problem on graphic hardware. In G. Yen, editor, *Proc. of the IEEE Conference on Evolutionary Computation (CEC), IEEE World Congress on Computational Intelligence (WCCI)*, pages 10345–10349, Vancouver, BC, Canada, July 16-21 2006. IEEE Press.

[190] G. Luque, E. Alba, and B. Dorronsoro. *Parallel Genetic Algorithms*, chapter 5, Parallel Metaheuristics: A New Class of Algorithms, pages 107–125. John Wiley & Sons, 2005.

[191] J. Lysgaard, A.N. Letchford, and R.W. Eglese. A new branch-and-cut algorithm for capacitated vehicle routing problems. *Mathematical Programming*, 100(2):423—-445, 2004.

[192] F.J. MacWilliams and N.J.A. Sloane. *The Theory of Error-Correcting Codes*. North-Holland, Amsterdam, 1977.

[193] J. Madera, E. Alba, and A. Ochoa. *Parallel Metaheuristics: A New Class of Algorithms*, chapter Parallel Estimation of Distribution Algorithms, pages 203–222. John Wiley & Sons, 2005.

[194] T. Mahnig and H. Mühlenbein. Comparing the adaptive Boltzmann selection schedule SDS to truncation selection. In *II Symposium on Artificial Intelligence. CIMAF99. Special Session on Distributions and Evolutionary Optimization*, pages 121–128, La Habana, 1999.

[195] B. Manderick and P. Spiessens. Fine-grained parallel genetic algorithm. In J.D. Schaffer, editor, *Proc. of the Third International Conference on Genetic Algorithms (ICGA)*, pages 428–433. Morgan Kaufmann, 1989.

[196] T. Maruyama, T. Hirose, and A. Konagaya. A fine-grained parallel genetic algorithm for distributed parallel systems. In *Proc. of the Fifth International Conference on Genetic Algorithms (ICGA)*, pages 184–190, San Francisco, CA, USA, 1993. Morgan Kaufmann.

[197] T. Maruyama, A. Konagaya, and K. Konishi. An asynchronous fine-grained parallel genetic algorithm. In *Proc. of the International Conference on Parallel Problem Solving from Nature II (PPSN-II)*, Lecture Notes in Computer Science (LNCS), pages 563–572. North-Holland, 1992.

[198] D. McAllester, B. Selman, and H. Kautz. Evidence for invariants in local search. In *National Conference on Artificial Intelligence*, pages 321–326, Providence, RI, 1997.

[199] G. Mendel. *Versuche über Pflanzen-Hybriden*. Verhandlungen des Naturforschedes Vereines in Brünn 4, 1865.

[200] D. Mester and O. Bräysy. Active guided evolution strategies for large-scale vehicle routing problems with time windows. *Computers & Operations Research*, 32:1593–1614, 2005.

[201] H. Mühlenbein. Parallel genetic algorithms, population genetic and combinatorial optimization. In *Proc. of the Third International Conference on Genetic Algorithms (ICGA)*, pages 416–421. Arlington, 1989.

[202] H. Mühlenbein. The equation for response to selection and its use for prediction. *Evolutionary Computation*, 5(3):303–346, 1997.

[203] H. Mühlenbein, M. Gorges-Schleuter, and O. Krämer. Evolution algorithms in combinatorial optimization. *Parallel Computing*, 7:65–88, 1988.

[204] H. Mühlenbein and G. Paab. From recombination of genes to the estimation of distributions I. Binary parameters. In H.M. Voigt, W. Ebeling, I. Rechenberg, and H.P. Schwefel, editors, *Proc. of the International Conference on Parallel Problem Solving from Nature IV (PPSN-IV)*, volume 1411 of *Lecture Notes in Computer Science (LNCS)*, pages 178–187. Springer-Verlag, Heidelberg, 1996.

[205] H. Mühlenbein and D. Schlierkamp-Voosen. The science of breeding and its application to the breeder genetic algorithm (BGA). *Evolutionary Computation*, 1(4):335–360, 1993.

[206] H. Mühlenbein, M. Schomish, and J. Born. The parallel genetic algorithm as a function optimizer. *Parallel Computing*, 17:619–632, 1991.

[207] H. Mühlenbein and H.M. Voigt. *Metaheuristics: Theory and Applications*, chapter Gene pool recombination in genetic algorithms, pages 53–62. Kluwer Academic Publishers, 1996.

[208] Z. Michalewicz. *Genetic Algorithms + Data Structure = Evolution Programs*. Springer-Verlag, Heidelberg, third edition, 1996.

[209] D.G. Mitchell, B. Selman, and H.J. Levesque. Hard and easy distributions for SAT problems. In P. Rosenbloom and P. Szolovits, editors, *Proc. of the Tenth National Conference on Artificial Intelligence*, pages 459–465, California, 1992. AAAI Press.

[210] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100, 1997.

[211] P. Moscato. *Handbook of Applied Optimization*, chapter Memetic Algorithms. Oxford University Press, 2000.

[212] T. Murata and M. Gen. Cellular genetic algorithm for multi-objective optimization. In *Proc. of the Fourth Asian Fuzzy System Symposium*, pages 538–542, 2002.

[213] T. Nakashima, T. Ariyama, and H. Ishibuchi. Combining multiple cellular genetic algorithms for efficient search. In *Proc. of the Asia-Pacific Conference on Simulated Evolution and Learning (SEAL)*, pages 712–716, 2002.

[214] T. Nakashima, T. Ariyama, T. Yoshida, and H. Ishibuchi. Performance evaluation of combined cellular genetic algorithms for function optimization problems. In *Proc. of the IEEE International Symposium on Computational Intelligence in Robotics and Automation*, pages 295–299, Kobe, Japan, July 16-20 2003. IEEE Press.

[215] A.J. Nebro, E. Alba, and F. Luna. Multi-objective optimization using grid computing. *Soft Computing*, page por aparecer, 2007.

[216] A.J. Nebro, J.J. Durillo, F. Luna, B. Dorronsoro, and E. Alba. A cellular genetic algorithm for multiobjective optimization. In D.A. Pelta and N. Krasnogor, editors, *Proceedings of the NICSO*, pages 25–36, Granada, Spain, 2006.

[217] A.J. Nebro, J.J. Durillo, F. Luna, B. Dorronsoro, and E. Alba. MOCell: A cellular genetic algorithm for multiobjective optimization. *International Journal of Intelligent Systems*, 2007. por aparecer.

[218] A.J. Nebro, J.J. Durillo, F. Luna, B. Dorronsoro, and E. Alba. A study of strategies for neigborhood replacement and archive feedback in a multiobjective cellular genetic algorithm. In *Proc. of the International Conference on Evolutionary Multi-criterion Optimization (EMO)*, Lecture Notes in Computer Science (LNCS). Springer-Verlag, Heidelberg, 2007. por aparecer.

[219] A.J. Nebro, F. Luna, E. Alba, A. Beham, and B. Dorronsoro. AbYSS: Adapting scatter search for multiobjective optimization. Technical Report ITI-2006-02, Dpto. de Lenguajes y CC.CC., Universidad de Málaga, 2006.

[220] N. Nedjah, E. Alba, and L. de Macedo Mourelle. *Parallel Evolutionary Computations*. Studies in Computational Intelligence. Springer-Verlag, Heidelberg, 2006.

[221] M.E.J. Newman. The structure and function of complex networks. *SIAM Review*, 45:167–256, 2003.

[222] S.-Y. Ni, Y.-C. Tseng, Y.-S. Chen, and J.-P. Sheu. The broadcast storm problem in a mobile ad hoc network. In *Proc. of the Annual ACM/IEEE International Conference on Mobile Computing and Networking*, pages 151–162, 1999.

[223] A. Ochoa, M. Soto, E. Alba, J. Madera, and B. Dorronsoro. Cellular estimation of distribution algorithms. *In preparation*, 2007.

[224] S. Olariu and A.Y. Zomaya. *Handbook of Bioinspired Algorithms and Applications*. Computer and Information Sience. Chapman & Hall/CRC, 2006.

[225] Y.S. Ong and A.J. Keane. Meta-lamarckian learning in memetic algorithms. *IEEE Transactions on Evolutionary Computation*, 8(2):99–110, April 2004.

[226] I.H. Osman. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problems. *Annals of Operations Research*, 41:421–451, 1993.

[227] J.L. Payne and M.J. Eppstein. Emergent mating topologies in spatially structured genetic algorithms. In *Proc. of the Genetic and Evolutionary Computation COnference (GECCO)*, pages 207–214, Seattle, Washington, USA, 2006. ACM Press.

[228] A. Pelc. *Handbook of Wireless Networks and Mobile Computing*, chapter Broadcasting In Wireless Networks, pages 509–528. John Wiley & Sons, 2002.

[229] M. Pelikan, D.E. Goldberg, and E. Cantú-Paz. BOA: The bayesian optimization algorithm. In *Proc. of the Genetic and Evolutionary Computation COnference (GECCO)*, pages 525–532. Morgan Kaufmann, 1999. Orlando, FL.

[230] M. Pelikan and H. Mühlenbein. The bivariate marginal distribution algorithm. In R. Roy, T. Furuhashi, and P. K. Chawdhry, editors, *Advances in Soft Computing-Engineering Design and Manufacturing*, pages 521–535. Springer-Verlag, Heidelberg, 1999.

[231] W. Peng and X.-C. Lu. On the reduction of broadcast redundancy in mobile ad hoc networks. In *Proc. of the ACM International Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC)*, pages 129–130. IEEE Press, 2000.

[232] F.B. Pereira, J. Tavares, P. Machado, and E. Costa. GVR: a new representation for the vehicle routing problem. In *Irish Conference Proceedings on Artificial Intelligence and Cognitive Science (AICS)*, pages 95–102, Ireland, 2002. Springer-Verlag, Heidelberg.

[233] C. Prins. A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers and Operations Research*, 31(12):1985–2002, May 2004.

[234] T.K. Ralphs, L. Kopman, W.R. Pulleyblank, and L.E. Trotter Jr. On the capacitated vehicle routing problem. *Mathematical Programming Series B*, 94:343–359, 2003.

[235] I. Rechenberg. Cybernetic solution path of an experimental problem. Technical report, Royal Aircraft Establishment, Library translation No. 1122, Farnborough, Hants., UK, 1965.

[236] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme und Prinzipien der biologischen Evolution*. Frommann-Holzboog, Stuttgart, 1973.

[237] G. Reinelt. TSPLIB: A travelling salesman problem library. *ORSA Journal on Computing*, 3:376–384 URL: http://www.iwr.uni–heidelberg.de/groups/comopt/software/TSPLIB95/, 1991.

[238] R. Reiter and A. Mackworth. A logical framework for depiction and image interpretation. *Applied Intelligence*, 41(3):123–155, 1989.

[239] J.L. Ribeiro-Filho, C. Alippi, and P. Treleaven. *Parallel Genetic Algorithms: Theory and Applications*, chapter Genetic algorithm programming environments, pages 65—-83. IOS Press, 1993.

[240] P. Rickers, R. Thomsen, and T. Krink. Applying self-organized criticality to the diffusion model. In D. Whitley, editor, *Proc. of the Genetic and Evolutionary Computation COnference (GECCO)*, pages 325–330, Las Vegas, Nevada, USA, 2000.

[241] G. Robertson. Parallel implementation of genetic algorithms in a classifier system. In *Proc. of the Second International Conference on Genetic Algorithms (ICGA)*, pages 140–147, 1987.

[242] Y. Rochat and E. Taillard. Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics*, 1:147–167, 1995.

[243] G. Rudolph. On takeover times in spatially structured populations: Array and ring. In K.K. Lai, O. Katai, M. Gen, and B. Lin, editors, *Proc. of the Second Asia-Pacific Conference on Genetic Algorithms and Applications (APGA)*, pages 144–151. Global-Link Publishing Company, 2000.

[244] G. Rudolph and J. Sprave. A cellular genetic algorithm with self–adjusting acceptance threshold. In *International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA)*, pages 365–372, 1995.

[245] R. Santana. Estimation of distribution algorithms with kikuchi approximations. *Evolutionary Computation*, 13(1):67–97, 2005.

[246] J. Sarma and K.A. De Jong. An analysis of the effect of the neighborhood size and shape on local selection algorithms. In H.M. Voigt, W. Ebeling, I. Rechenberg, and H.P. Schwefel, editors, *Proc. of the International Conference on Parallel Problem Solving from Nature IV (PPSN-IV)*, volume 1141 of *Lecture Notes in Computer Science (LNCS)*, pages 236–244. Springer-Verlag, Heidelberg, 1996.

[247] J. Sarma and K.A. De Jong. An analysis of local selection algorithms in a spatially structured evolutionary algorithm. In T. Bäck, editor, *Proc. of the Seventh International Conference on Genetic Algorithms (ICGA)*, pages 181–186. Morgan Kaufmann, 1997.

[248] J.D. Schaffer and L.J. Eshelman. On crossover as an evolutionary viable strategy. In R.K. Belew and L.B. Booker, editors, *Proc. of the Fourth International Conference on Genetic Algorithms (ICGA)*, pages 61–68. Morgan Kaufmann, 1991.

[249] D. Schlierkamp-Voosen and H. Mühlenbein. Adaption of population sizes by competing subpopulations. In *Proc. IEEE International Conference on Evolutionary Computation (CEC)*, pages 330–335, Piscataway, NY, 1996. IEEE Press.

[250] B. Schönfisch and A. de Roos. Synchronous and asynchronous updating in cellular automata. *BioSystems*, 51:123–143, 1999.

[251] H.-P. Schwefel. *Kybernetische Evolution als Strategie der Experimentellen Forschung in der Strömungstechnik*. PhD thesis, Technical University of Berlin, 1965.

[252] H.-P. Schwefel. *Numerical Optimization of Computer Models*. John Wiley & Sons, Chichester, England, 1981.

[253] B. Selman, H. Kautz, and B. Cohen. Noise strategies for improving local search. In *National Conference on Artificial Intelligence*, pages 337–343, California, 1994. AAAI Press.

[254] D. Simoncini, S. Verel, P. Collard, and M. Clergue. Anisotropic selection in cellular genetic algorithms. In *Proc. of the Genetic and Evolutionary Computation COnference (GECCO)*, pages 559–566, Seattle, Washington, USA, 2006. ACM Press.

[255] M. Sipper. *Evolution of Parallel Cellular Machines: The Cellular Programming Approach*. Lecture Notes in Computer Science (LNCS). Springer-Verlag, Heidelberg, 1997.

[256] J.E. Smith and F. Vavak. Replacement strategies in steady state genetic algorithms: static environments. In Banzhaf and Reeves, editors, *Foundations of Genetic Algorithms V (FOGA)*, pages 219–234. Morgan Kaufmann, 1998.

[257] M. Soto, A. Ochoa, S. Acid, and L.M. de Campos. Introducing the polytree aproximation of distribution algorithm. In *II Symposium on Artificial Intelligence. CIMAF99. Special Session on Distributions and Evolutionary Optimization*, pages 360–367, 1999. Cuba.

[258] P. Spiessens and B. Manderick. A massively parallel genetic algorithm: Implementation and first analysis. In R.K. Belew and L.B. Booker, editors, *Proc. of the Fourth International Conference on Genetic Algorithms (ICGA)*, pages 279–286. Morgan Kaufmann, 1991.

[259] J. Sprave. Linear neighborhood evolution strategies. In A.V. Sebald and L.J. Fogel, editors, *Proc. of the Annual Conference on Evolutionary Programming*, pages 42–51, River Edge, NJ, USA, 1994. World Scientific.

[260] J. Sprave. A unified model of non-panmictic population structures in evolutionary algorithms. In P.J Angeline, Z. Michalewicz, M. Schoenauer, X. Yao, and A. Zalzala, editors, *Proc. IEEE International Conference on Evolutionary Computation (CEC)*, volume 2. IEEE Press, 1999.

[261] J. Stender. *Parallel Genetic Algorithms: Theory and Applications*. IOS Press, Amsterdam, The Netherlands, 1993.

[262] D.R. Stinson. *An Introduction to the Design and Analysis of Algorithms*. The Charles Babbage Research Center, Winnipeg, Manitoba, Canada, 1985 (second edition, 1987).

[263] I. Stojmenovic and J. Wu. *Mobile Ad Hoc Networking*, chapter Broadcasting and activity scheduling in ad hoc networks, pages 205–229. Wiley-IEEE Press, 2004.

[264] R. Storn and K. Price. Differential evolution—a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical Report TR-95-012, International Computer Science Institute, Berkeley, CA, 1995.

[265] P.D. Surry and N.J. Radcliffe. RPL2: A language and parallel framework for evolutionary computing. In Y. Davidor, H.-P. Schwefel, and R. Männer, editors, *Proc. of the International Conference on Parallel Problem Solving from Nature III (PPSN-III)*, pages 628–637, Berlin, 1994. Springer-Verlag, Heidelberg.

[266] E. Taillard. Parallel iterative search methods for vehicle-routing problems. *Networks*, 23(8):661–673, 1993.

[267] E.-G. Talbi. *Parallel Combinatorial Optimization.* John Wiley & Sons, 2006.

[268] E.-G. Talbi and P. Bessière. A parallel genetic algorithm for the graph partitioning problem. In *Proc. of the International Conference on Supercomputing*, pages 312–320. ACM Press, 1991.

[269] R. Tanese. Distributed genetic algorithms. In J.D. Schaffer, editor, *Proc. of the Third International Conference on Genetic Algorithms (ICGA)*, pages 434–439. Morgan Kaufmann, 1989.

[270] C.D. Tarantilis and C.T. Kiranoudis. BoneRoute: an adaptive memory-based method for effective fleet management. *Annals of Operations Research*, 115:227–241, 2002.

[271] R. Thomsen, P. Rickers, and T. Krink. A religion-based spatial model for evolutionary algorithms. In H.-P. Schwefel, M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, and J.J. Merelo, editors, *Proc. of the International Conference on Parallel Problem Solving from Nature VI (PPSN-VI)*, Paris, France, 2000. Springer-Verlag, Heidelberg.

[272] M. Tomassini. The parallel genetic cellular automata: Application to global function optimization. In R.F. Albrecht, C.R. Reeves, and N.C. Steele, editors, *Proc. of the International Conference on Artificial Neural Networks and Genetic Algorithms*, pages 385–391. Springer-Verlag, Heidelberg, 1993.

[273] M. Tomassini. *Spatially Structured Evolutionary Algorithms: Artificial Evolution in Space and Time.* Natural Computing Series. Springer-Verlag, Heidelberg, 2005.

[274] A. Töorn and Ž. Antanas. *Global Optimization*, volume 350 of *Lecture Notes in Computer Science (LNCS).* Springer-Verlag, Heidelberg, Berlin, Germany, 1989.

[275] P. Toth and D. Vigo. *The Vehicle Routing Problem.* Monographs on Discrete Mathematics and Applications. SIAM, Philadelphia, 2001.

[276] P. Toth and D. Vigo. The granular tabu search and its application to the vehicle routing problem. *INFORMS Journal on Computing*, 15(4):333–346, 2003.

[277] S. Tsutsui and Y. Fujimoto. Forking genetic algorithm with blocking and shrinking modes. In S. Forrest, editor, *Proc. of the Fifth International Conference on Genetic Algorithms (ICGA)*, pages 206–213, San Mateo, CA, 1993. Morgan Kaufmann.

[278] S. Tsutsui, A. Ghosh, D. Corne, and Y. Fujimoto. A real coded genetic algorithm with an explorer and exploiter populations. In T. Bäck, editor, *Proc. of the Seventh International Conference on Genetic Algorithms (ICGA)*, pages 238–245. Morgan Kaufmann, 1997.

[279] A. Van Breedam. *An Analysis of the Behavior of Heuristics for the Vehicle Routing Problem for a Selection of Problems with Vehicle Related, Customer-related, and Time-related Constraints.* PhD thesis, University of Antwerp - RUCA, Belgium, 1994.

[280] A. Van Breedam. Comparing descent heuristics and metaheuristics for the vehicle routing problem. *Computers & Operations Research*, 28:289–315, 2001.

[281] A. Van Breedam. A parametric analysis of heuristics for the vehicle routing problem with side-constraints. *European Journal of Operations Research*, 137:348–370, 2002.

[282] D.A. Van Veldhuizen and G.B. Lamont. Multiobjective evolutionary algorithm research: A history and analysis. Technical Report TR-98-03, Dept. Elec. Comput. Eng., Air Force Inst. Technol., Wright-Patterson, AFB, OH, 1998.

[283] H.-M. Voigt, H. Mühlenbein, and D. Cetković. Fuzzy recombination for the breeder genetic algorithm. In L. Eshelman, editor, *Proc. of the Sixth International Conference on Genetic Algorithms (ICGA)*, pages 104–111, 1995.

[284] H. M. Voigt, I. Santibáñez-Koref, and J. Born. Hierarchically structured distributed genetic algorithms. In R. Männer and B. Manderick, editors, *Proc. of the International Conference on Parallel Problem Solving from Nature II (PPSN-II)*, pages 155–164, Amsterdan, 1992. North-Holland.

[285] K. Weinert, J. Mehnen, and G. Rudolph. Dynamic neighborhood structures in parallel evolution strategies. *Complex Systems*, 2004. to appear.

[286] W. Whewell and R.E. Butts. *Theory of Scientific Method.* Hackett Pub Co Inc, 2nd edition, 1989.

[287] P.M. White and C.C. Pettey. Double selection vs. single selection in diffusion model gas. In S. Forrest, editor, *Proc. of the Fifth International Conference on Genetic Algorithms (ICGA)*, pages 174–180. Morgan Kaufmann, 1993.

[288] D. Whitley. Cellular genetic algorithms. In S. Forrest, editor, *Proc. of the Fifth International Conference on Genetic Algorithms (ICGA)*, page 658, California, CA, USA, 1993. Morgan Kaufmann.

[289] D. Whitley, R. Beveridge, C. Graves, and K. Mathias. Test driving three 1995 genetic algorithms: New test functions and geometric matching. *Journal of Heuristics*, 1:77–104, 1995.

[290] D. Whitley, S. Rana, J. Dzubera, and K.E. Mathias. Evaluating evolutionary algorithms. *Applied Intelligence*, 85:245–276, 1997.

[291] D. Whitley, T. Starkweather, and D. Fuquay. Scheduling problems and traveling salesman: The genetic edge recombination operator. In J.D. Schaffer, editor, *Proc. of the Third International Conference on Genetic Algorithms (ICGA)*, pages 133–140. Morgan Kaufmann, 1989.

[292] B. Williams and T. Camp. Comparison of broadcasting techniques for mobile ad hoc networks. In *Proc. of the ACM International Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC)*, pages 194–205. ACM Press, 2002.

[293] S. Wolfram. *Theory and Applocations of Cellular Automata*. World Scientific, Singapore, 1986.

[294] D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, April 1997.

[295] S. Wright. Isolation by distance. *Genetics*, 28:114–138, 1943.

[296] J. Wu and W. Lou. Forward-node-set-based broadcast in clustered mobile ad hoc networks. *Wireless Communications and Mobile Computing*, 3(2):155–173, 2003.

[297] F. Xhafa. A cellular memetic algorithm for resource allocation in grid systems. In E.-G. Talbi and L. Jourdan, editors, *Proceedings of the META 2006*, 2006. CD edition.

[298] F. Xhafa, E. Alba, and B. Dorronsoro. Efficient batch job scheduling in grids using cellular memetic algorithms. In *Nature Inspired Distributed Computing (NIDISC) sessions of the International Parallel and Distributed Processing Simposium (IPDPS) Workshop*. IEEE Press, 2007. Por aparecer.

[299] T. Yu. *An Analysis of the Impact of Functional Programming Techniques on Genetic Programming*. PhD thesis, University College London, 1999.

[300] E. Zitzler, K. Deb, and L. Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *IEEE Transactions on Evolutionary Computation*, 8(2):173–195, 2000.

[301] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the strength pareto evolutionary algorithm. Technical Report 103, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH), 2001.

[302] E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation*, 23(4):257–271, 1999.

Índice de Términos

AF, 86
AF+PH, 86
aislamiento por distancia, 17
alelo, 22
Algoritmo
  con rejilla adaptativa, 219
  de Estimación de Distribución, 117
  Evolutivo (EA), 1, 13
  Genético Celular (cGA), 178
  Genético Meta-celular, 140
  Memético, 127
  Memético Celular, 128
algoritmos miembros, 120
auto-adaptación, 80

búsqueda local
  λ-Intercambio, 184, 189
  2-Opt, 184, 189
BMDA, 119
BOA, 119

cambio de ratio, 83
cDFCNT, 217, 224
cEA
  asíncrono, 20
  síncrono, 20
celda, 120
cGA, 22
  adaptativo, 79
  jerárquico (HcGA), 105
  multi-objetivo, 151
  para optimización numérica, 203
  paralelo, 139
cMOGA, 152, 212, 217
comportamiento
  proactivo, 215
  reactivo, 215
cromosoma, 22
crowding, 217
cubrimiento de conjuntos, 227

depredador-presa, modelo, 151
DFCN, 211
  densidad de seguridad, 216
DFCNT, 217
dispersión, 156

EA
  celular, 15
  de difusión, 28
  de grano fino, 28
  de individuos paralelos, 28
  de polinización de plantas, 28
  de selección local, 28
  descentralizado, 15
  distribuido, 15
  Heterogéneo, 16
  masivamente paralelo, 28
ECGA, 119
EDAs
  celulares, 117
  clasificación, 119
  con dependencias entre pares de variables, 119
  con dependencias múltiples, 119
  sin dependencias, 119
EGNA_{BGe}, 119
ESaM, 156

frente de Pareto, 151
función de vecindad, 41

GA
  celular, 22
  generacional, 186

gen, 22
generación, 20
GRAD, 129
grafos de mundo pequeño, 31
grid, 139

Hamming, 243
hibridación, 127
hipergrafo, 30
hipervolumen, 227

JCell, 165
    configuración, 170
jMetal, 152

métrica
    cubrimiento de conjuntos, 48
    dispersión (Δ), 47, 157, 160, 163
    distancia generacional (GD), 47, 156, 157, 160
    hipervolumen (HV), 47, 159, 160, 163
    Número de óptimos de Pareto, 48
MANETs, 211
    áreas de alta densidad, 212
    dispositivo, 211
    Metropolitana, 211
    nodo, 211
    simulación de entorno de autopista, 214, 221
    simulación de entorno de centro comercial, 213, 220
    simulación de entorno metropolitano, 213, 221
MEA, 151
metaheurística, 12
MIMIC, 119
MIMIC_C, 119
MOCell, 152, 153, 158
MOP, 151
mutación
    combinada, 183
    de cláusulas no satisfechas (UCM), 132
    polinómica, 219
    por inserción, 183
    por intercambio, 183
    por inversión, 183

nicho, 17, 32, 97
NSGA-II, 151, 154, 160

optimización, 10
    con restricciones, 11
    multi-objetivo, 151
    numérica, 203

PADA, 119
PAES, 151
Pareto, 151
PBIL, 119
PH, 86
presión de selección, 52
problema
    ef_{10}, 249
    Ackley, 249
    ConstrEx, 155, 253
    COUNTSAT, 243
    de Ajuste Polinomial (Chebischev), 252
    de de optimización combinatoria, 243
    de Ecuaciones Lineales (Sle), 251
    de las Tareas de Mínima Espera (MTTP), 247
    de Llenado de Recipiente (BPP), 180
    de Modulación en Frecuencia de Sonidos (FMS), 245, 251
    de optimización binaria, 10
    de optimización continua, 10, 249
    de optimización entera, 10
    de optimización heterogénea, 10
    de Rutas de Vehículos (VRP), 177, 179
    de Rutas de Vehículos con Capacidad limitada (CVRP), 143, 178, 179
    de Satisfacibilidad (SAT), 127, 141, 248
    del Diseño de Códigos Correctores de Errores (ECC), 244
    del Máximo Corte de un Grafo (MAXCUT), 245
    Fonseca, 155, 253
    Fractal, 249
    Griewangk, 249
    IsoPeak, 245
    Kursawe, 155, 253
    múltiple del viajante de comercio (MTSP), 180
    Masivamente Multimodal (MMDP), 246
    OneMax, 247
    Osyczka2, 155, 253
    P-PEAKS, 248
    Plateau, 248

Rastrigin, 249
Rosenbrock, 249
Schaffer, 155, 253
Schwefel, 249
Sphere, 249
Srinivas, 155, 253
Tanaka, 155, 253
WFG, 159, 253
ZDT, 155, 159, 253

radio, 21
ratio, 21, 30
adaptativa, 84
pre-programada, 83
recombinación
BLX-*alpha*, 204
de cláusulas no satisfechas (UCR), 132
EFR, 206
ERX, 182
FCB, 206
genérica, 144
SBX, 219
Retraso de Estimación Aleatoria (RAD), 215

SA, 130
SAT, adaptación de pesos paso a paso, 249
selección
anisótropa, 31
por disimilitud, 108
por ordenación lineal (LR), 63
por ruleta (RW), 62
por torneo binario (BT), 52
significancia estadística, 46
Sistema Adaptativo, 39
Granulado, 40
SPEA2, 151, 154, 160
speedup, 141

Test
t-test, 46
ANOVA, 46
de Kolmogorov-Smirnov, 46
de Kruskal-Wallis, 46
tiempo de takeover, 30, 52

UMDA, 119

vecindario, 17
de Moore, 18
de Von Newmann, 18
en espacio de soluciones, 11
NEWS, 18
ventana de observación, 214

WSAT, 130