UNIVERSIDAD
DE MÁLAGA

Algoritmos Meméticos con Propiedades Self-⋆ para la Optimización de Problemas Complejos

Tesis presentada por Rafael Nogueras Sánchez
bajo la supervisión del Dr. Carlos Cotta Porras
para obtener el grado de Doctor por la Universidad de Málaga

2015

Programa Tecnologías de la Información
Departamento de Lenguajes y Ciencias de la Computación
Escuela Técnica Superior de Ingeniería Informática
Universidad de Málaga

AUTOR: Rafael Nogueras Sánchez
http://orcid.org/0000-0002-2456-5925

EDITA: Publicaciones y Divulgación Científica. Universidad de Málaga

D. **Carlos Cotta Porras**, Profesor Titular de Universidad del Departamento de Lenguajes y Ciencias de la Computación de la Universidad de Málaga

CERTIFICA

que D. **Rafael Nogueras Sánchez**, 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

Algoritmos Meméticos con Propiedades Self-⋆ para la Optimización de 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, 22 de Julio de 2015

Firmado: Carlos Cotta Porras
Profesor Titular de Universidad
Dpto. Lenguajes y Ciencias de la Computación
Universidad de Málaga

Agradecimientos

Esta tesis doctoral ha supuesto un enorme esfuerzo para muchas personas a las que quiero agradecer su compromiso y atención, en unas facetas o en otras, lo que ha permitido que pudiera llegar a su conclusión.

En primer lugar a mi mujer Mamen y a mi hijo Javier, por su apoyo constante, su tolerancia y su confianza absoluta. Siempre estaré en deuda con ambos por el tiempo que les he robado. También debo expresar mi gratitud a mis padres, por su completa entrega y dedicación, tanto en este como en otros capítulos de mi formación académica. Y no puedo olvidarme de mi hermano Rubén, Ingeniero en Informática como yo, que siempre ha sabido resolver todas mis dudas técnicas y me ha dado muy acertados consejos.

Finalmente, mi reconocimiento y admiración más sinceros para el director de este trabajo, el Dr. Carlos Cotta. Quiero agradecerle su paciencia y disponibilidad, mayores de lo que podría haber esperado, así como su enorme capacidad docente. Quiero también darle las gracias por haberme transmitido sus conocimientos y su entusiasmo por lo que hace, dos pilares básicos para el desarrollo de esta tesis. Además, ha sabido enseñarme a investigar, a que siempre hay otra pregunta que se podría formular y otra hipótesis que se podría plantear. Es justo reconocer que este trabajo nunca habría podido finalizarse sin su ayuda plena. Carlos es sin duda alguna, el mejor ejemplo a seguir y el mejor modelo a imitar.

Índice de contenidos

Índice de acrónimos

ACO Optimización con Colonia de Hormigas.

AI Inteligencia Artificial.

ANN Red Neuronal Artificial.

CO Optimización Combinatoria.

COMIT Combinación de Optimizadores con Árboles de Información Mutua.

COP Problemas de Optimización Combinatoria.

CPU Unidad Central de Proceso.

DE Evolución Diferencial.

EA Algoritmo Evolutivo.

EC Computación Evolutiva.

EDA Algoritmo de Estimación de Distribuciones.

EP Programación Evolutiva.

ES Estrategia Evolutiva.

FPTAS Esquema de Aproximación en Tiempo Completamente Polinómico.

GA Algoritmo Genético.

GLS Búsqueda Local Guiada.

GP Programación Genética.

GPU Unidad de Procesamiento Gráfico.

GRASP Procedimiento Adaptativo de Búsqueda Voraz Aleatorizada.

HC Algoritmo de Escalada.

ILS Búsqueda Local Iterativa.

LAN Red de Área Local.

MA Algoritmo Memético.

MC Computación Memética.

MIMIC Maximización de la Información Mutua para Agrupamiento de la Entrada.

MMA Algoritmo Multimemético.

MT Máquina de Turing.

MTD Máquina de Turing Determinista.

MTND Máquina de Turing No Determinista.

NFLT Teorema de No Free Lunch.

P2P Peer-to-Peer.

PBIL Aprendizaje Incremental Basado en Población.

PC Complejidad Parametrizada.

PSD Densidad Espectral de Potencia.

PSO Optimización por Enjambre de Partículas.

PTAS Esquema de Aproximación en Tiempo Polinómico.

QAP Problema de la Asignación Cuadrática.

QRTD Distribución Cualificada de las Ejecuciones.

SA Recocido Simulado.

SAT Problema de Satisfacibilidad Lógica.

SLS Búsqueda Local Estocástica.

SS Búsqueda Dispersa.

TS Búsqueda Tabú.

TSP Problema del Viajante de Comercio.

UMDA Algoritmo de Distribución Marginal Univariable.

VC Recubrimiento de Vértices.

VNS Búsqueda en Vecindades Variables.

Índice de algoritmos

Índice de tablas

Índice de figuras

Capítulo 1

Introducción

Esta tesis trata sobre el estudio de los algoritmos multimeméticos como metaheurística utilizada para resolver problemas de optimización combinatoria en entornos distribuidos inestables con topologías de red complejas y recursos volátiles, utilizando propiedades self-$\star$ para mitigar los efectos negativos producidos por la inestabilidad de dichos entornos.

1.1. Planteamiento

La creciente complejidad de los problemas de optimización actuales (la cual evoluciona frecuentemente a un ritmo más rápido que las prestaciones hardware de los sistemas disponibles) requiere desarrollar técnicas algorítmicas cada vez más sofisticadas. La teoría de la complejidad ha sido de gran ayuda en este sentido, ya que ha permitido clasificar los problemas computacionales en clases de complejidad en función de los recursos necesarios (normalmente tiempo y espacio) para poder resolverlos. Esto a su vez ha permitido definir el concepto de tratabilidad, entendiéndose que un problema es tratable si se puede resolver en un tiempo y espacio razonables. Lamentablemente existen múltiples problemas interesantes computacionalmente que son intratables desde diferentes perspectivas.

Para resolver los problemas mencionados en el párrafo anterior se ha venido empleando durante las últimas décadas las metaheurísticas, esto es, procedimientos de búsqueda y optimización de propósito general para resolver problemas computacionales (normalmente problemas de optimización combinatoria). Existen múltiples metaheurísticas diferentes y muchas clasificaciones posibles, pero entre todas ellas son las de índole bioinspirado las que son de interés para esta tesis, y más concretamente los algoritmos evolutivos. Estos están basados en los principios fundamentales de la Teoría de la Evolución de las Especies y aplican estos conceptos en la resolución de diversos problemas de optimización. Los algoritmos evolutivos han sido durante muchos años una herramienta poderosa para afrontar este tipo de problemas. Sin embargo, el teorema No Free Lunch introdujo limitaciones teóricas al uso de algoritmos de búsqueda y optimización de

propósito general y generó la necesidad de incluir conocimiento específico de los problemas en la propia mecánica lógica de los algoritmos. Esto indujo el uso de nuevas técnicas cooperativas que unieran de alguna manera las técnicas habituales de propósito general con nuevos mecanismos para incorporar conocimiento a las mismas.

Como respuesta a las limitaciones descritas anteriormente surgieron nuevos modelos como los algoritmos meméticos, los cuales son una familia de técnicas de optimización basadas en población (al igual que los algoritmos evolutivos) que combinan ideas de diferentes metaheurísticas para explotar este conocimiento específico del problema con la idea de obtener mejores resultados tanto desde el punto de vista de la eficiencia como de la calidad de las soluciones obtenidas. El funcionamiento de estas técnicas está inspirado en el concepto de meme, entendiéndose este como un fragmento de conocimiento que se va adquiriendo y moldeando a partir de la experiencia y cuya interpretación más habitual en el marco de la computación memética es utilizarlo como proceso de búsqueda local. Por otro lado, los algoritmos multimeméticos son una subclase de los algoritmos meméticos en la que los memes se incorporan de forma explícita a los genotipos y evolucionan a la vez que estos. Todo esto introduce una complejidad adicional en los algoritmos meméticos con respecto a los algoritmos evolutivos clásicos, lo que provoca que su diseño y funcionamiento sean más complejos.

Debido a la complejidad anteriormente mencionada es necesario utilizar nuevos conceptos teóricos para dotar a los algoritmos de capacidades de auto-gestión y auto-adaptación con el objetivo de mitigar el cada vez mayor esfuerzo necesario para su diseño y parametrización, trasladando parte de este al propio algoritmo y limitando por lo tanto la intervención humana. De esta forma surge la computación autónoma y las propiedades self-$\star$, las cuales son un conjunto de propiedades que debe tener (al menos alguna de ellas) un sistema para considerarlo autónomo desde alguna perspectiva. Estas propiedades se pueden aplicar tanto para el ajuste dinámico de los parámetros propios del algoritmo evolutivo como para cualquier otro aspecto del mismo, como por ejemplo la topología subyacente en el caso de algoritmos distribuidos. Según lo anterior, se dispone de un algoritmo evolutivo con conocimiento específico del problema y propiedades self-$\star$. Sin embargo, hay que considerar también las cuestiones de rendimiento. En la actualidad los problemas son cada vez más grandes y complejos lo que requiere no solo de algoritmos bien diseñados, con una parametrización adecuada y que sean autónomos, sino también que sean eficientes y tengan un buen rendimiento.

El desarrollo de plataformas de computación distribuida durante las últimas décadas ha permitido mejorar las prestaciones para aquellos algoritmos que sean capaces de ejecutarse en este tipo de entornos, lo que requiere ciertos cambios en el diseño de los mismos. Una de las grandes ventajas de los algoritmos evolutivos en general es precisamente su fácil adaptación de un entorno secuencial a uno distribuido, lo que los convierte en un medio muy interesante para afrontar los retos del presente (y del futuro) en términos de resolución de problemas eficientemente bajo plataformas distribuidas. En esta línea, los algoritmos evolutivos basados en

islas son un buen modelo de referencia distribuido sobre el que pueden aplicarse diferentes extensiones que lo doten de capacidades adicionales de tolerancia a fallos y auto-gestión.

1.2. Objetivos

Los principales objetivos del trabajo se esquematizan a continuación:

- Estudiar algoritmos multimeméticos (algoritmos meméticos con autogeneración) para aplicarlos a la resolución de problemas de optimización combinatoria.
- Desarrollar algoritmos multimeméticos con propiedades self-⋆ que sean capaces de auto-adaptarse a escenarios cambiantes.
- Realizar diseños con estructura espacial que aborden los objetivos anteriores sobre entornos inestables manteniendo el rendimiento.

Para conseguir alcanzar los objetivos anteriores se ha utilizado una metodología por capas, es decir, comenzando por el diseño de un algoritmo multimemético sencillo se ha llegado a un algoritmo multimemético con estructura espacial y propiedades self-⋆ incorporadas, añadiendo sucesivamente nuevas características al algoritmo fundamental. Concretamente, se parte de un algoritmo panmíctico (sin estructura espacial) que no dispone de propiedades self-⋆ (excepto la propia autogeneración) para sucesivamente añadirle primero la capacidad de auto-adaptación de los memes y posteriormente la incorporación de modelos probabilísticos para la creación de los descendientes. En una capa posterior, se dota al algoritmo de características distribuidas, extendiendo el algoritmo inicial a un modelo basado en islas que se ejecutan en nodos diferentes. Una vez se dispone de una versión estructurada en islas, se le añaden propiedades de tolerancia a fallos en entornos inestables y de auto-escalabilidad mediante mecanismos de auto-equilibrado de carga entre los nodos. Finalmente, se añaden propiedades de auto-reparación y auto-curación, bien a través del auto-muestreo para obtener nuevos individuos cuando se requieren reinicializaciones de los mismos, o bien mediante el autocableado para reconexionar de forma dinámica la estructura de la red subyacente cuando se producen desconexiones por la caída de algunos nodos del sistema.

1.3. Estructura de la Tesis

Esta tesis doctoral se ha organizado en varios capítulos que parten desde lo más elemental hasta los modelos más complejos desarrollados. El Capítulo 2 está formado por los conceptos generales necesarios para poder abordar la lectura del resto de la tesis. Por ello se han dedicado algunas secciones a la teoría de la complejidad computacional y a los problemas de optimización combinatoria,

a las metaheurísticas en general y a los algoritmos evolutivos y meméticos en particular. Posteriormente se aborda en este mismo capítulo la descripción de las denominadas propiedades self-⋆, para finalizar con un repaso por los conceptos fundamentales relacionados con la computación paralela.

En el Capítulo 3 se tratan en profundidad los algoritmos multimeméticos, se analiza cómo se propagan los memes, se muestran resultados experimentales sobre la auto-adaptación memética y se presenta un modelo híbrido de algoritmo multimemético que utiliza modelos probabilísticos para generar la población de descendientes. Los capítulos 4 y 5 están dedicados a los algoritmos multimeméticos distribuidos utilizando un modelo basado en islas. En el capítulo 4 se estudian todos los aspectos involucrados en el diseño de un algoritmo multimemético basado en islas, como las políticas de migración entre islas o algunas estrategias de tolerancia a fallos para gestionar la inestabilidad de los nodos. Sobre una de ellas se profundiza al final del capítulo, concretamente sobre la estrategia denominada checkpoint, que consiste en crear puntos de restauración periódicos con la información necesaria para recuperar un nodo que previamente ha fallado, y finalmente, en el Capítulo 5 se emplean diversas propiedades self-⋆ para hacer que los algoritmos sean robustos en escenarios de alta volatilidad. Concretamente, se utilizan propiedades de auto-escalado para reequilibrar el tamaño de cada nodo disponible cuando otros abandonan el sistema y por lo tanto se pierden individuos, de auto-muestreo para que la recuperación de los nodos se base en un modelo probabilístico de los individuos existentes y de auto-reparación para reconstruir la topología de la red, ya que conforme van saliendo y entrando nodos en el sistema esta se va deformando y perdiendo su aspecto original.

En el Capítulo 6 se discuten las conclusiones finales sobre el trabajo realizado y se explican algunas de las posibles líneas de investigación que se pueden abordar en el futuro relacionadas con esta tesis. Por último se incluyen dos apéndices: el Apéndice A explica de forma detallada los problemas de test utilizados durante los experimentos realizados en todos los capítulos y el Apéndice B contiene algunas tablas con resultados numéricos correspondientes a los resultados experimentales de los capítulos 4 y 5.

1.4. Contribuciones

Las principales aportaciones de esta tesis doctoral al campo de la computación memética se resumen a continuación:

- Diseño de algoritmos meméticos con auto-generación.
  - Análisis de la propagación de memes en un algoritmo multimemético.
  - Diseño de un modelo híbrido compuesto por un algoritmo multimemético y algoritmos de estimación de distribuciones para modelar la población de descendientes.

- Análisis y diseño de algoritmos multimemeticos distribuidos sobre entornos inestables.
  - Análisis de diferentes estrategias de migración en el contexto de los algoritmos multimemeticos basados en islas.
  - Análisis y diseño de estrategias de tolerancia a fallos en entornos altamente volátiles.
- Incorporación de propiedades self-$\star$ a los algoritmos multimemeticos distribuidos.
  - Análisis y diseño de propiedades de auto-escalabilidad a través del auto-equilibrado de carga entre los nodos de un algoritmo multimemético distribuido.
  - Utilización de modelos probabilísticos para implementar el auto-muestreo de poblaciones en nodos que se reincorporan a la topología subyacente.
  - Análisis y diseño de propiedades de auto-reparación mediante la reconexión dinámica de los nodos de la red.

1.5. Listado de Publicaciones

Durante esta tesis doctoral se han publicado los siguientes artículos que sustentan el contenido de la misma:

- **Revistas**

R1 R. Nogueras, C. Cotta, A Study on Meme Propagation in Multimemetic Algorithms, *International Journal of Applied Mathematics and Computer Science*, 25(3), 2015 [276][1].

R2 R. Nogueras, C. Cotta, Studying Fault-tolerance in Island-based Evolutionary and Multimemetic Algorithms, *Journal of Grid Computing*, 2015, DOI: 10.1007/s10723-014-9315-6 [277][2].

R3 R. Nogueras, C. Cotta, Studying Self-balancing Strategies in Island-based Multimemetic Algorithms, *Journal of Computational and Applied Mathematics*, 2015, DOI: 10.1016/j.cam.2015.03.047 [278][3].

- **Capítulos de libros**

CL1 R. Nogueras, C. Cotta, Towards Resilient Multimemetic Systems on Unstable Networks with Complex Topology, *Advances in Evolutionary Algorithm Research*, G. Papa (ed.), Nova Science Pub., 2015 [en edición] [279].

[1] IF=1.227, Q2 – MATHEMATICS, APPLIED
[2] IF=1.507, Q1 – COMPUTER SCIENCE, THEORY & METHODS
[3] IF=1.266, Q1 – MATHEMATICS, APPLIED

- **Congresos**

  C1 R. Nogueras, C. Cotta, Analyzing Meme Propagation in Multimemetic Algorithms: Initial Investigations, *Proceedings of the 2013 Federated Conference on Computer Science and Information Systems*, pp. 1013-1019, Sep 2-6, IEEE Press, Cracow (Poland), 2013 [269].

  C2 R. Nogueras, C. Cotta, On Meme Self-Adaptation in Spatially-Structured Multimemetic Algorithms, *Numerical Methods and Applications - 8th International Conference*, Ivan Dimov, Stefka Fidanova, Ivan Lirkov (Eds.), pp. 70-77, Lecture Notes in Computer Science 8962, Springer-Verlag, Berlin Heidelberg, 2014 [271].

  C3 R. Nogueras, C. Cotta, A Study on Multimemetic Estimation of Distribution Algorithms, *Parallel Problem Solving from Nature - PPSN XIII*, T. Bartz-Beielstein et al. (eds.), pp. 322-331, Lecture Notes in Computer Science 8672, Springer-Verlag, Berlin Heidelberg, 2014 [272].

  C4 R. Nogueras, C. Cotta, An Analysis of Migration Strategies in Island-Based Multimemetic Algorithms, *Parallel Problem Solving from Nature - PPSN XIII*, T. Bartz-Beielstein et al. (eds.), pp. 731-740, Lecture Notes in Computer Science 8672, Springer-Verlag, Berlin Heidelberg, 2014 [270].

  C5 R. Nogueras, C. Cotta, Sensitivity Analysis of Checkpointing Strategies for Multimemetic Algorithms on Dynamic Complex Networks, *10th International Conference on Large Scale Scientific Computations*, Lecture Notes in Computer Science, Springer-Verlag, Berlin Heidelberg, 2015 [en edición] [275].

  C6 R. Nogueras, C. Cotta, Self-Sampling Strategies for Multimemetic Algorithms in Unstable Computational Environments, *Bioinspired Computation in Artificial Systems*, J.M. Ferrández Vicente at al. (eds.), pp. 69-78, Lecture Notes in Computer Science 9108, Springer-Verlag, Berlin Heidelberg, 2015 [274].

  C7 R. Nogueras, C. Cotta, Self-Balancing Multimemetic Algorithms in Dynamic Scale-Free Networks, *Applications of Evolutionary Computing*, A.M. Mora, G. Squillero (eds.), pp. 177-188, Lecture Notes in Computer Science 9028, Springer-Verlag, Berlin Heidelberg, 2015 [273].

De manera más precisa, el análisis de algoritmos multimeméticos presentado en el Capítulo 3 está sustentado por R1, C1, C2 y C3. Por su parte, el despliegue de estas técnicas en un modelo de islas y el estudio de su funcionamiento en entornos inestables presentado en el capítulo 4 se apoya en R2, C4 y C5. Finalmente, el empleo de propiedades self-⋆ avanzadas como auto-escalado y auto-reparación descritas en el capítulo 5 se sustenta en R3, CL1, C6 y C7.

Capítulo 2

Antecedentes y Conceptos Previos

Este capítulo tiene como objetivo fundamental presentar un punto de vista general sobre todos los aspectos y técnicas que tienen relación con el contenido de la tesis. Dado que la misma se centra en el estudio de diferentes aspectos del funcionamiento de diversas técnicas de optimización, en primer lugar se expondrán los principales conceptos relacionados con la teoría de la complejidad computacional y con la resolución de problemas de optimización combinatoria (Sección 2.1). La gran dificultad asociada intrínsecamente con ciertas clases de problemas motiva el empleo de metaheurísticas, por lo que se procederá acto seguido a presentar las ideas básicas de estas así como una clasificación de dichas técnicas (Sección 2.2). En particular se pondrá el foco de atención en metaheurísticas de corte bioinspirado, y más concretamente evolutivo, que tratan de aprender y utilizar los principios que rigen la evolución de las especies para aplicarlos a la resolución de problemas (Sección 2.3). Sin embargo, las limitaciones teóricas impuestas a las técnicas que no utilizan conocimiento específico del problema hace necesario introducir mecanismos de hibridación para mejorar la calidad de las soluciones encontradas, utilizando para ello la computación memética (Sección 2.4). Finalmente se muestra la dificultad del diseño y funcionamiento que tienen estos algoritmos y cómo se puede solventar a través de mecanismos de auto-adaptación, definiendo para ello las denominadas propiedades self-$\star$, que permiten reducir la intervención humana en el diseño de los algoritmos (Sección 2.5), y por último, dada la complejidad creciente de los problemas del mundo real se trata también la necesidad de mejorar el rendimiento de las metaheurísticas en general y de los algoritmos evolutivos y meméticos en particular mediante el diseño de algoritmos que introduzcan paralelismo (Sección 2.6).

2.1. La Complejidad de Resolver Problemas

Desde la Antigüedad el hombre ha sentido la necesidad de dar respuesta a las múltiples preguntas que se planteaba. Inicialmente eran cuestiones que hoy en día consideraríamos muy elementales, pero con el paso del tiempo estas preguntas fueron incrementando su complejidad. Con el bagaje formal y las herramientas proporcionadas primero por las Matemáticas y posteriormente por las Ciencias de la Computación, es posible abordar de manera rigurosa dicha complejidad, esto es, el coste computacional asociado a la resolución de un problema.

En esta sección se van a explicar los principales conceptos relacionados con la teoría de la complejidad y la clasificación de los problemas computacionales en clases de complejidad.

2.1.1. Problemas y Optimización

Esta subsección se centra en los problemas computacionales, concretamente en su definición y clasificación, pasando a continuación a estudiar un caso particular de estos, como son los Problemas de Optimización Combinatoria (COP)[1].

2.1.1.1. Problemas Computacionales

Desde el punto de vista de las ciencias de la computación, un *problema computacional* $\mathcal{P}$ se puede definir como una relación entre un conjunto de instancias $I_{\mathcal{P}}$ propio del problema y un conjunto de soluciones $S_{\mathcal{P}}$. A cada instancia $x \in I_{\mathcal{P}}$ se le asocia un conjunto de soluciones $S_{\mathcal{P}}(x)$, que denota el conjunto de soluciones posibles para dicha instancia del problema $\mathcal{P}$ [24, 141, 155, 257, 292]. El conjunto $S_{\mathcal{P}}(x)$ se conoce también como el conjunto de soluciones válidas o aceptables.

El objetivo es diseñar un algoritmo A que resuelva cualquier instancia $x \in I_{\mathcal{P}}$ del problema $\mathcal{P}$. Dependiendo del tipo de respuesta esperada, esto es, de qué entendemos por resolver el problema, los problemas computacionales se pueden clasificar en:

- *Problemas de decisión*: son aquellos que admiten como respuesta únicamente "sí" o "no", es decir, para una instancia $x \in I_{\mathcal{P}}$, la cuestión es decidir si existe una solución $y \in S_{\mathcal{P}}(x)$.

- *Problemas de conteo*: el objetivo en este caso es calcular el número de soluciones existentes para una instancia concreta $x \in I_{\mathcal{P}}$ de un problema $\mathcal{P}$, es decir, determinar la cardinalidad del conjunto $S_{\mathcal{P}}(x)$.

- *Problemas de enumeración*: son aquellos cuyo objetivo es encontrar el conjunto de todas las posibles soluciones $S_{\mathcal{P}}(x)$ del problema considerado para una instancia $x \in I_{\mathcal{P}}$ particular.

[1] *Combinatorial Optimization Problems*. Por consistencia con la literatura se van a utilizar siempre acrónimos en inglés

- *Problemas de satisfacción*: en este tipo de problemas el objetivo es hallar una solución del conjunto $S_{\mathcal{P}}(x)$ de soluciones válidas para la instancia $x \in I_{\mathcal{P}}$ del problema $\mathcal{P}$.

- *Problemas de Optimización*: este tipo de problemas son similares a los de satisfacción con la diferencia de que existe un criterio para comparar soluciones en base al cual estas se pueden ordenar de mejor a peor. Típicamente, en estos problemas se dispone de una función objetivo f definida como $f : S_{\mathcal{P}}(x) \to \mathbb{N}$ que proporciona la calidad o el coste de las soluciones, de forma que el problema consiste en encontrar la solución $s \in S_{\mathcal{P}}(x)$ que maximice o minimice la función f.

En las siguientes secciones se trata con más detalle este último tipo de problemas, más concretamente los de optimización combinatoria.

2.1.1.2. Optimización Combinatoria

La Optimización Combinatoria (CO)[2] es una rama de la optimización que se caracteriza principalmente porque su dominio se compone de problemas en los que el conjunto de posibles soluciones es discreto o se puede reducir a un conjunto discreto. Más concretamente, un COP consiste en un problema del que se pretende encontrar la mejor solución entre todas las posibles. Según Papadimitriou y Steiglitz [293] en los COP se busca un objeto de un conjunto finito –o posiblemente infinito numerable. Este objeto normalmente suele ser un número entero, un subconjunto, una permutación o una estructura algebraica del tipo de un árbol o un grafo.

De manera más precisa se puede definir un COP como sigue [45]:

Definición 1. *Un problema de optimización combinatoria $\mathcal{P} = \langle X, D, f, R\rangle$ se define como:*

- *Un conjunto de variables $X = \{x_1, \ldots, x_n\}$;*

- *Un conjunto de dominios $D = \{D_1, \ldots, D_n\}$ para cada una de las variables anteriores;*

- *Una función objetivo $f : D_1 \times \ldots \times D_n \to \mathbb{N}$;*

- *Un conjunto de restricciones R entre el conjunto de variables X.*

El conjunto $S \subseteq D_1 \times \ldots \times D_n$ que cumple el conjunto de restricciones R se considera el espacio de búsqueda del problema y cada elemento $s \in S$ es una solución candidata para el problema $\mathcal{P}$. Dado un problema de optimización combinatoria $\mathcal{P}$ el objetivo es encontrar una solución $s^* \in S$ que minimice (o que maximice; asumimos minimización sin pérdida de generalidad) la función objetivo

[2] *Combinatorial Optimization*

f, es decir, $\forall s \in S : f(s^*) \leqslant f(s)$. En este contexto, s^* se denomina un óptimo global de $\mathcal{P}$. Nótese que puede haber más de una solución que sea óptimo global.

Algunos ejemplos de COP son el Problema del Viajante de Comercio (TSP)[3], el Problema de la Asignación Cuadrática (QAP)[4] o el problema de la planificación de tareas, por citar algunos ejemplos. Debido a que este tipo de problemas tiene una importancia elevada para la industria y en el campo científico se han desarrollado numerosos algoritmos para resolverlos. No todos estos algoritmos son equivalentes, ni con respecto a la calidad de las soluciones que proporcionan ni con respecto a su coste computacional. En general, ambos aspectos están relacionados, así como condicionados por la dificultad intrínseca del problema que se desea resolver. La siguiente sección estudia este factor.

2.1.2. Complejidad Computacional

La Teoría de la Complejidad trata sobre el estudio de la complejidad de los problemas computacionales –esto es, de los recursos necesarios para resolver un problema determinado y la clasificación de tales problemas en clases de complejidad de acuerdo a dicho consumo de recursos. Estos recursos suelen ser tiempo y espacio [24, 69, 155, 292], aunque en ocasiones se puede recurrir a otros. En cualquier caso, el consumo se define como una función del tamaño de la entrada del problema. La Teoría de la Complejidad formaliza matemáticamente lo anterior. Para ello se suele utilizar la notación $\mathcal{O}(\cdot)$, conocida también como notación de Landau, definida formalmente de la siguiente manera:

Definición 2. *Sean dos funciones $f, g : \mathbb{R} \to \mathbb{R}$. Se dice que $f(n) \in \mathcal{O}(g(n))$ si y solo si $\exists n_0, c > 0$, tales que $|f(n)| \leqslant c|g(n)|$ para $n > n_0$, es decir, $f(n)$ está acotada por $g(n)$ para valores grandes de n.*

Es importante distinguir entre analizar el coste computacional de un algoritmo para un problema concreto y el mínimo coste de todos los posibles algoritmos que se puedan diseñar e implementar para resolver este mismo problema. Es esto último lo que permite establecer una clasificación de los problemas en función del coste necesario para su resolución.

Antes de profundizar en esta clasificación es necesario describir el modelo computacional en base al cual se considera el coste de espacio y/o tiempo. Esto se va a realizar a través del concepto de Máquina de Turing (MT). Una MT [349–351] es un modelo matemático de un computador que permite formalizar la noción de algoritmo. Existen diferentes definiciones equivalentes de MT. Considérese la siguiente:

Definición 3. *Una máquina de Turing con una sola cinta puede definirse como una tupla $M = (Q, q_0, \Sigma, \delta)$ donde:*

[3] *Travelling Salesman Problem*
[4] *Quadratic Assignment Problem*

- Q *es un conjunto finito de estados.*
- $q_0 \in Q$ *es el estado inicial.*
- Σ *es un conjunto finito de símbolos denominado alfabeto.*
- $\delta : Q \times \Sigma \rightarrow \{h\} \cup (Q \times \Sigma \times \{I, D\})$ *es una función de transición de estados, donde el estado* h *representa el estado de parada y los símbolos* I *y* D*, representan respectivamente, un movimiento a la izquierda o a la derecha.*

Una máquina de Turing [171, 363] puede visualizarse como un dispositivo que dispone de una cinta infinita dividida en celdas sobre la que se realizan operaciones de lectura y escritura a través de un cabezal, usando símbolos de un alfabeto definido y siguiendo un movimiento determinado por una función de transición (δ), que lleva la máquina de un estado a otro a partir de un estado inicial (q_0) y alcanzando eventualmente el estado de parada (h). La máquina lee una celda de la cinta en cada paso, borrando el símbolo en el que se encuentra y escribiendo un nuevo símbolo sobre la misma. A continuación se desplaza el cabezal una posición a la izquierda o a la derecha según la función de transición, hasta que se alcance el estado de parada, siendo el contenido de la cinta en ese momento la salida de la máquina.

Consideremos Γ_i el contenido de la cinta de una MT en un instante t_i. Se entiende por computación en una MT la secuencia de pares $(q, \Gamma) \in Q \times \Sigma^*$ desde t_0 (q_0, Γ_0) hasta t_h (q_h, Γ_h), siendo $q_h = h$ –el estado de parada de la MT– y t_h el instante de tiempo correspondiente. En este caso Γ_0 es la entrada de la MT y Γ_h su salida.

Cuando solo existe una transición posible para cada par $(q, \sigma) \in Q \times \Sigma^*$ en la función de transición, nos hallamos ante una Máquina de Turing Determinista (MTD), en cambio, si existe más de una transición para algún par entonces se trata de una Máquina de Turing No Determinista (MTND), que se puede definir como sigue:

Definición 4. *Una máquina de Turing no determinista con una sola cinta puede definirse como una tupla* $M = (Q, q_0, \Sigma, \Delta)$ *donde:*

- Q *es un conjunto finito de estados.*
- $q_0 \in Q$ *es el estado inicial.*
- Σ *es un conjunto finito de símbolos denominado alfabeto.*
- $\Delta : Q \times \Sigma \rightarrow 2^{[\{h\} \cup (Q \times \Sigma \times \{I, D\})]}$ *es una función de transición de estados, donde el estado* h *representa el estado de parada y los símbolos* I *y* D *representan, respectivamente, un movimiento a la izquierda o a la derecha.*

Si en un par $(q, \sigma) \in Q \times \Sigma^*$ no hay ninguna acción posible, la MTD se bloquea y se para, en cambio si hay más de una, la MTD se bifurca y todas las ramas se exploran en paralelo. La computación puede en este caso representarse como un árbol cuyos nodos son pares $(q, \Gamma) \in Q \times \Sigma^*$. Este paralelismo no proporciona ventajas en términos de qué problemas se pueden resolver, pero sí permite una reducción del tiempo de cómputo.

2.1.3. Clases de Complejidad

Una clase de complejidad es un conjunto de problemas con una complejidad computacional similar. Estas clases se pueden definir en función de ciertos factores:

- El tipo de problema computacional: en la mayoría de las ocasiones se trata de problemas de decisión, aunque las clases de complejidad se pueden definir en función de cualquiera de los problemas enumerados en la sección previa.
- El modelo de computación: lo más habitual es utilizar una MTD o una MTND, pero hay clases de complejidad basadas en otros "artefactos" como circuitos lógicos o más recientemente máquinas de Turing cuánticas [290].
- Los recursos computacionales: los recursos tienen que ser acotados y establecerse unas cotas para cada recurso. Estas dos características se fusionan normalmente, hablando de conceptos tales como "tiempo polinómico", "espacio exponencial", etc.

A la luz de lo anterior, se pueden considerar por ejemplo clases del tipo **TIME**$(f(n))$ y **NTIME**$(f(n))$, correspondientes a los problemas de decisión cuya resolución requiere de un tiempo $\mathcal{O}(f(n))$ en una MTD y una MTND respectivamente, siendo n el tamaño de la entrada. Del mismo modo podemos definir las clases **SPACE**$(f(n))$ y **NSPACE**$(f(n))$ basadas en este caso en el espacio necesario para la resolución del problema considerado.

2.1.3.1. Reducción, Dureza y Completitud

Uno de los conceptos fundamentales en teoría de la complejidad es el de problema completo para una clase **C**. Para definir este concepto es preciso establecer previamente la idea de reducción polinómica:

Definición 5. *Sean $\mathcal{P}_1$ y $\mathcal{P}_2$ dos problemas de decisión. Una reducción polinómica de $\mathcal{P}_1$ en $\mathcal{P}_2$ es una función $\rho : I_{\mathcal{P}_1} \to I_{\mathcal{P}_2}$ que satisface las siguientes condiciones:*

1. *ρ requiere un tiempo $\mathcal{O}(p(n))$ para ejecutarse, donde $p(n)$ es una función polinómica de n.*
2. *Para toda instancia $x \in I_{\mathcal{P}_1}$, x tiene como respuesta "sí" para $\mathcal{P}_1$ si y solo si $\rho(x)$ tiene como respuesta "si" también para $\mathcal{P}_2$.*

Básicamente una reducción polinómica puede entenderse como un mecanismo eficiente para transformar un problema en otro. Ahora se puede definir un problema duro para una clase como sigue:

Definición 6. *Un problema de decisión $\mathcal{P}$ es $\boldsymbol{C}$-duro cuando para todo $\mathcal{P}' \in \boldsymbol{C}$ existe una reducción polinómica de $\mathcal{P}'$ en $\mathcal{P}$.*

Un problema $\mathcal{P}$ que sea **C**-duro es por lo tanto un problema al menos tan difícil de resolver como todos los problemas en **C** ya que si se dispusiera de un algoritmo para $\mathcal{P}$ se podría resolver cualquier problema en **C**. Finalmente, podemos definir la noción de problema completo para una clase como:

Definición 7. *Un problema de decisión $\mathcal{P}$ es $\boldsymbol{C}$-completo cuando se satisfacen las siguientes condiciones:*

1. $\mathcal{P} \in \boldsymbol{C}$.
2. $\mathcal{P}$ *es* $\boldsymbol{C}$*-duro.*

Los problemas completos para una clase son importantes ya que caracterizan la dificultad de resolución de la misma.

2.1.3.2. Clases P y NP

Para muchos problemas de decisión se pueden diseñar algoritmos que, en el caso peor, pueden encontrar el óptimo global en un tiempo acotado por una función polinómica del tamaño de la entrada considerada. La clase de problemas que se pueden resolver por una máquina determinista en tiempo polinómico se dice que pertenecen a la clase **P**:

Definición 8. *Un problema pertenece a la clase $\boldsymbol{P}$ si y solo si una MTD puede resolverlo en tiempo polinómico.*

Como parece evidente, estos serían los problemas resolubles más fácilmente –eficientemente–, aunque si la constante multiplicativa de la función polinómica o su grado son grandes en la práctica podrían resultar muy costosos de resolverse para instancias de un tamaño elevado. La otra clase fundamental es la denominada clase **NP**, que corresponde a los problemas de decisión que se pueden resolver en tiempo polinómico por una MTND, es decir, una máquina hipotética capaz de "adivinar" las decisiones correctas a la hora de resolverlo.

Definición 9. *Un problema se encuentra en la clase $\boldsymbol{NP}$ si y solo si una MTND puede resolverlo en tiempo polinómico.*

Es evidente que de las definiciones anteriores se deduce que $\mathbf{P} = \mathbf{TIME}(p(n))$ y $\mathbf{NP} = \mathbf{NTIME}(p(n))$. Por consiguiente, $\mathbf{P} \subseteq \mathbf{NP}$, aunque sigue siendo una cuestión abierta si un problema que puede ser resuelto por una máquina no determinista en tiempo polinómico podría también resolverlo una máquina determinista en tiempo polinómico, es decir, si $\mathbf{P} = \mathbf{NP}$.

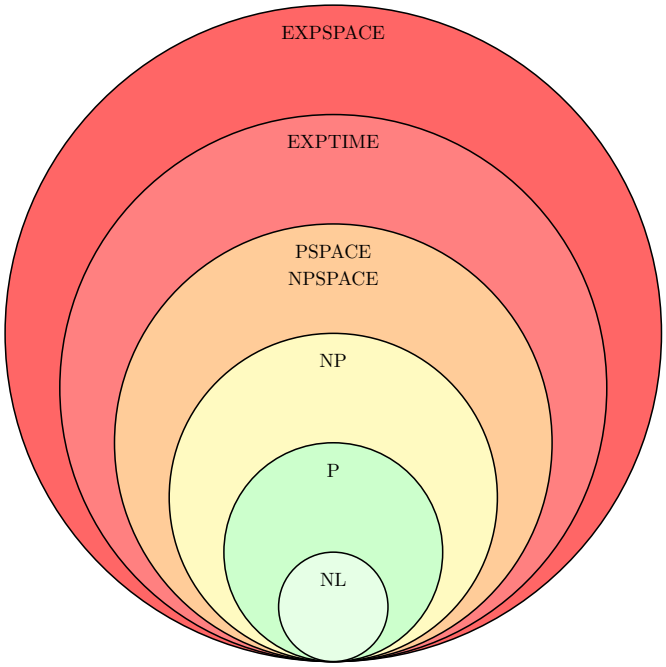


Figura 2.1: Esquema con algunas de las principales clases de complejidad. Nótese que **NL** = **SPACE**($\log n$), **(N)PSPACE** =**(N)SPACE**($p(n)$), **EXPTIME** = **TIME**($2^{p(n)}$) y **EXPSPACE** = **SPACE**($2^{p(n)}$).

La teoría de la **NP**-completitud tiene su origen en el trabajo de Cook [68] en 1971, en el que se probó que el Problema de Satisfacibilidad Lógica (SAT)[5] de lógica matemática es **NP**-completo. El resultado se conoce como "Teorema de Cook". A partir de aquí se empezaron a presentar problemas que podían considerarse **NP**-completos en múltiples campos, como la combinatoria, la lógica, la teoría de grafos y el álgebra en general.

Dentro de la clase **NP** son especialmente importantes los problemas **NP**-completos. No se conoce ningún algoritmo polinómico para resolver un problema **NP**-completo ya que, por definición, si se encontrara dicho algoritmo para un solo problema entonces todo problema en **NP** sería también polinómico, y por lo tanto, **P** = **NP** [155, 178, 196, 292]. Esta es una cuestión trascendental ya que **P** se asocia a los problemas que son tratables, es decir, eficientemente resolubles.

Existen muchas otras clases de complejidad, algunas de las cuales se ilustran en la Fig. 2.1, caracterizando diferentes niveles de dificultad/coste para la resolución de los correspondientes problemas.

2.1.3.3. Esquemas de Aproximabilidad

Los problemas de optimización **NP**-duros son lo suficientemente complejos para no permitir encontrar una solución exacta en tiempo polinómico. En estos

[5] *Boolean Satisfiability Problem*

casos se puede recurrir a la búsqueda de una solución aproximada de la óptima. Es importante asumir que si se pretende resolver un problema **NP**-duro con un algoritmo "eficiente" –en tiempo polinómico–, este no siempre será capaz de encontrar la solución óptima, pero sí puede encontrar una solución con un coste menor [50, 51, 99, 102, 141]. En este contexto, es importante considerar el ratio de rendimiento para un cierto algoritmo.

Definición 10. *Dada una instancia $x \in I_{\mathcal{P}}$ de un problema de optimización $\mathcal{P}$ y un algoritmo A que lo resuelve, el ratio de rendimiento $R(A, x)$ del algoritmo A para la instancia $x \in I_{\mathcal{P}}$ se define como:*

$$R(A, x) = \text{máx}\left(\frac{q(A(x))}{q(opt(x))}, \frac{q(opt(x))}{q(A(x))}\right) \tag{2.1}$$

donde $A(x)$ es la solución que encuentra un algoritmo A para la instancia x, $opt(x)$ es la solución óptima de la instancia x y la función $q(s) : S_{\mathcal{P}}(x) \to \mathbb{N}$ devuelve la calidad de la solución s.

Obviamente, tanto en un problema de maximización como en uno de minimización, si el algoritmo de aproximación A encuentra la solución óptima entonces $R = 1$ y si la solución que encuentra está lejos del óptimo el valor de R es arbitrariamente grande.

Definición 11. *Dado un problema de optimización $\mathcal{P}$ y un algoritmo de aproximación A para $\mathcal{P}$, entonces A es una algoritmo de aproximación-γ para $\mathcal{P}$ si dada cualquier instancia $x \in I_{\mathcal{P}}$ de $\mathcal{P}$, el ratio de rendimiento R de la solución aproximada s proporcionada por A está siempre acotado por γ, es decir: $R(A, x) \leqslant \gamma$*

De la definición anterior se puede extraer que si $\gamma = 1$ el algoritmo de aproximación es un algoritmo exacto. Considérense ahora las siguientes clases:

Definición 12. *La clase **NPO** corresponde a todos los problemas de optimización cuya versión de decisión pertenece a la clase **NP**.*

Definición 13. *La clase **APX** es una subclase de **NPO** para la que existen algoritmos de aproximación con un ratio de rendimiento constante, es decir, existe un algoritmo en tiempo polinómico que garantiza encontrar una solución con una aproximación a la solución óptima determinada mediante un factor constante.*

Se pueden diseñar algoritmos que sean capaces de encontrar soluciones cada vez mejores en detrimento de sus tiempos de ejecución. El objetivo es buscar un compromiso entre la calidad de las soluciones alcanzadas y el tiempo de computación del algoritmo diseñado para llegar a esas soluciones. Para este propósito se ideó el concepto de esquema de aproximación:

Definición 14. *Un Esquema de Aproximación en Tiempo Polinómico (PTAS)*[6] *es un esquema de aproximación para un problema $\mathcal{P}$ tal que para cualquier $\varepsilon > 0$,*

[6] *Polynomial-Time Approximation Scheme*

existe un algoritmo en ***TIME***(n^k) *que permite encontrar una solución cuyo coste es menor que* $1+\varepsilon$ *veces el coste de la solución óptima.*

Un PTAS permite aproximar "eficientemente" la solución óptima de un problema. Nótese sin embargo que el exponente del polinomio k puede depender de ε, por lo que aproximaciones buenas pueden ser muy costosas. Es por ello que se define el concepto de Esquema de Aproximación en Tiempo Completamente Polinómico (FPTAS)[7].

Definición 15. *Un FPTAS es un esquema de aproximación para un problema* $\mathcal{P}$ *tal que para cualquier* $\varepsilon > 0$*, existe un algoritmo polinómico de grado independiente de* ε *que permite encontrar una solución cuyo coste es menor que* $1+\varepsilon$ *veces el coste de la solución óptima.*

Por extensión, pueden definirse las clases **PTAS** y **FPTAS** como las clases de problemas que admiten uno de estos esquemas de aproximación. Las clases anteriores cumplen la siguiente regla de inclusión $\mathbf{FPTAS} \subseteq \mathbf{PTAS} \subseteq \mathbf{APX} \subseteq \mathbf{NPO}$. Estas inclusiones serían estrictas si y solo si $\mathbf{P} \neq \mathbf{NP}$.

2.1.3.4. Complejidad Parametrizada

La Complejidad Parametrizada (PC)[8], inicialmente desarrollada por Downey y Fellows [100, 101, 111] se ocupa de la clasificación de los problemas computacionales con respecto a uno o varios parámetros de entrada [122], en vez de a uno solo (el tamaño en bits de la entrada) como lo hace la teoría de la complejidad clásica. Esto permite una clasificación con una granularidad más fina, sobre todo para aquellos problemas **NP**-duros. Es importante mencionar que las clasificaciones clásicas basadas en las clases **P** y **NP** siempre tienen en cuenta el peor escenario posible. Por ejemplo, el conocido problema SAT es, en general, resoluble fácilmente para la mayoría de las instancias generadas de forma aleatoria, en cambio, para instancias que se encuentran en una fase de transición entre la satisfacibilidad y la no satisfacibilidad su resolución resulta altamente costosa [143].

El paradigma PC establece una jerarquía de clases de complejidad parametrizada $\mathbf{FPT} \subseteq \mathbf{W}[1] \subseteq \mathbf{W}[2] \subseteq \ldots \subseteq \mathbf{W}[t] \subseteq \mathbf{W}[SAT] \subseteq \mathbf{W}[P]$ que permite discriminar problemas en diferentes clases de complejidad en función del parámetro elegido. Por ejemplo, los problemas de la clase **FPT**[9] tienen algoritmos cuya complejidad del peor caso es $\mathcal{O}(f(k)n^c)$, donde k es el parámetro considerado, $f(k)$ es una función arbitraria de k y c es una constante independiente de k. Por el contrario, la complejidad para resolver problemas de la clase $\mathbf{W}[1]$ es $\mathcal{O}(f(k)n^{g(k)})$, la cual es considerablemente más dura en general. Considérese también que si

[7] *Fully Polynomial-Time Approximation Scheme*
[8] *Parameterized Complexity*
[9] *Fixed-parameter tractable*

un problema tiene un FPTAS entonces está en **FPT** (aunque el recíproco no es necesariamente cierto).

Como se menciona en el párrafo anterior, la jerarquía **W** [62, 101] permite encapsular problemas de creciente dificultad donde la pertenencia de un problema concreto a una clase se establece a través de circuitos lógicos. Estos circuitos tienen como entrada una posible solución y producen como salida un valor lógico que indica si la solución es apropiada para el problema considerado. Evidentemente el diseño de cada circuito depende del problema y la complejidad de este determina a qué clase pertenece dicho problema. En el diseño del circuito un parámetro importante a considerar es el número máximo de puertas lógicas cuyo "fan-in"[10] es ilimitado (no acotado por una función solo de k) en un camino desde la entrada hasta la salida. Este parámetro se denomina en la literatura anglosajona "*weft*" (trama) y cuanto mayor es, mayor es la clase a la que pertenece el circuito considerado.

Existen un gran número de problemas que requieren un coste superior al polinómico para resolverlos cuando se mide su complejidad en términos únicamente del tamaño de la entrada, pero teniendo en cuenta un segundo parámetro k podrían tratarse de forma razonable, bajo ciertas condiciones de k, como que su crecimiento/disminución fueran relativamente pequeños. Para los problemas **NP**-completos o **NP**-duros es poco probable que existan algoritmos eficientes independientes del parámetro de entrada, ya que hasta ahora todos los algoritmos conocidos para resolver este tipo de problemas requieren un tiempo superior al polinómico con respecto al tamaño de la entrada. En cambio, si consideramos un algoritmo que sea solo exponencial con respecto a un parámetro fijo, en este caso hay problemas que sí se pueden resolver en un tiempo razonable para valores pequeños del parámetro fijo. Según Downey, Fellows y Stege [102] aproximadamente la mitad de los problemas parametrizados **NP**-completos están en **FPT**, como por ejemplo, el conocido problema del Recubrimiento de Vértices (VC)[11] que se puede definir como sigue:

Definición 16. *Dado un grafo $G = (V, E)$ y un entero positivo k, ¿existe un subconjunto $V' \subseteq V$ de tamaño máximo k tal que para cada arista $(u, v) \in E$ se cumpla que $\{u, v\} \cap V' \neq \emptyset$?*

Papadimitriou y Yannakakis [291] demostraron que VC se puede resolver en $\mathcal{O}(3^k n)$, donde n es el número de vértices de G y Chen, Kanj y Jia [59] propusieron un algoritmo para resolverlo en tiempo $\mathcal{O}(1{,}278^k + n)$. Esto quiere decir que este problema se puede resolver en tiempo lineal para un número arbitrario de vértices si k es fijo. El problema VC no es un caso aislado, ya que existen muchos problemas **NP**-duros para los que se puede crear un algoritmo que los resuelva en un tiempo limitado. Por contra, los problemas que se encuentran en alguna clase superior de la jerarquía son mucho más complejos de resolver. La clase **W**[1] sirve como medida en la actualidad de la "intratabilidad" de los problemas.

[10] número de entradas de la puerta
[11] *Vertex Cover*

2.2. Metaheurísticas

En las secciones anteriores se ha introducido el concepto de tratabilidad, se ha considerado que un problema es intratable si no pertenece a la clase **P** y se ha visto que no todos los problemas existentes son tratables. Posteriormente se ha determinado que se puede establecer una granularidad más fina para la clase **NP** a través de la complejidad parametrizada y de las clases **FPT**. Por otro lado es bien conocido que existen múltiples COPs para los que no existe un algoritmo en tiempo polinómico o incluso aunque exista, en la práctica su tiempo de ejecución es demasiado elevado para instancias muy grandes cuando el exponente del polinomio tiene un valor alto.

Un ejemplo de estos problemas es el Problema del Clique Máximo[12], que consiste en encontrar el subgrafo completo (clique) de tamaño máximo. A continuación se define formalmente la versión decisional de este problema:

Definición 17. *Dado un grafo $G = (V, E)$ y un entero positivo k, ¿existe un subconjunto $V' \subseteq V$ de tamaño k o superior tal que $\forall u, w \in V'$, $(u, w) \in E$?*

Este problema, como otros muchos de la teoría de grafos, surge a partir del modelado del mundo real. Por ejemplo, una red social se puede definir mediante un grafo, donde los vértices representan a los individuos y los ejes las conexiones entre ellos. Encontrar el máximo clique equivale a encontrar el mayor subconjunto de personas que se conocen todas entre sí. Esto se puede resolver mediante "fuerza bruta", es decir, verificando cada uno de los posibles subgrafos existentes, pero es evidente que el consumo de recursos necesarios puede ser enorme conforme aumenta tanto el número de personas como el tamaño del clique buscado, lo que lo hace inviable. Más concretamente, este es un problema **W**[1]-completo para esta parametrización y por lo tanto, intratable en general (no es resoluble en tiempo polinómico, ni aproximable mediante un FPTAS).

Existen otros muchos problemas que no son **FPT**, como por ejemplo el Problema del Empaquetado de Conjuntos[13] [26] que es **W**[1]-completo o el de Selección de Características[14] [74] que es **W**[2]-completo por citar solo algunos ejemplos (pueden encontrarse otros muchos problemas en [56, 88, 168]). Las heurísticas surgen para solucionar los inconvenientes de las técnicas completas (exactas o aproximadas) para resolver este tipo de problemas, ya que sacrifican el objetivo, concretamente la garantía de encontrar o aproximar la solución óptima a cambio de encontrar a menudo soluciones de buena calidad con un coste razonable.

2.2.1. Concepto

Las metaheurísticas son plantillas algorítmicas de propósito general para resolver problemas. El término "metaheurística" lo introdujo inicialmente Glover

[12] *Maximum Clique Problem*
[13] *Set Packing*
[14] *Feature Selection*

en [149] y tiene su origen en la fusión de dos palabras griegas. "Heurística" se deriva del verbo *heuriskein* que significa "encontrar" y el sufijo "meta" significa "después de", "por encima de" o "más allá de". Antes de que la comunidad científica adoptase definitivamente este término se utilizaba también "heurísticas modernas" [307]. Según Osman y Laporte [287]:

> Una metaheurística se define formalmente como un proceso de generación iterativo que dirige una heurística subordinada combinando inteligentemente diferentes conceptos para explorar y explotar el espacio de búsqueda utilizando estrategias de aprendizaje para estructurar la información con el objetivo de encontrar soluciones cuasi-óptimas de forma eficiente.

Inicialmente la definición de *metaheurística* estaba ligada a aquellos métodos capaces de aunar procedimientos de búsqueda local (una definición precisa de los cuales se proporcionará más adelante) con otras estrategias de más alto nivel con el objetivo de encontrar las mejores soluciones a un problema considerado, explorando el espacio de búsqueda y evitando estancarse en soluciones subóptimas. El origen del término metaheurística se encuentra ligado a las disciplinas de la Inteligencia Artificial (AI)[15] y la Investigación Operativa [45, 151, 307] y generalmente se refiere a algoritmos heurísticos para optimización.

A menudo suele ser habitual establecer la distinción entre *métodos constructivos* y *métodos iterativos*. Los primeros producen soluciones desde cero, añadiendo componentes a una solución parcial inicial, hasta encontrar la solución definitiva. Estos algoritmos generalmente son más rápidos aunque suelen encontrar soluciones de peor calidad que los algoritmos iterativos. Por otro lado, estos últimos parten de una solución inicial (o conjunto de ellas) y de forma iterativa las van transformando en otras de mejor calidad basándose en una relación de vecindad que permite pasar de una solución a la siguiente. Una definición formal de esta relación es:

Definición 18. *Una relación de vecindad es una función* $\mathcal{N} : S \to 2^S$*, siendo* S *el espacio de búsqueda, de tal forma que a cada* $s \in S$ *asigna un conjunto de vecinos* $\mathcal{N}(s) \subseteq S$*.* $\mathcal{N}(s)$ *se denomina vecindario o vecindad de* s*.*

A partir de esta relación se pueden definir de forma más precisa dos conceptos fundamentales en el contexto de las metaheurísticas, como son el de óptimo local y óptimo global:

Definición 19. *Una solución* s *es un óptimo local si tiene mejor calidad que todos sus vecinos, es decir,* $\forall s' \in \mathcal{N}(s), q(s) \leqslant q(s')$ *en caso de minimización.*

Definición 20. *Una solución* s *es un óptimo global si y solo si no hay otra solución de mejor calidad, es decir,* $\forall s' \in S, q(s) \leqslant q(s')$ *en caso de minimización.*

[15] *Artificial Intelligence*

Con estas definiciones se puede definir el concepto de búsqueda local como la acción encaminada a encontrar una solución $s' \in \mathcal{N}(s)$ que cumpla que $q(s') \leqslant q(s)$.

Una metaheurística aglutina un conjunto de conceptos que se pueden utilizar para definir métodos heurísticos que permiten resolver una amplia gama de problemas. Dicho de otro modo, una metaheurística es un enfoque algorítmico de propósito general que se puede aplicar a diferentes problemas de optimización. Esto es lo que hace que estas técnicas sean mecanismos sumamente útiles para abordar la resolución de una gran variedad de problemas con un tiempo de desarrollo considerablemente menor que si se tuviese que diseñar un algoritmo específico para cada una de ellas.

Blum y Roli [45] definen un conjunto de propiedades características de las metaheurísticas:

- Las metaheurísticas son estrategias que dirigen el proceso de búsqueda.
- El objetivo es explorar eficientemente el espacio de búsqueda para encontrar una solución cuasi-óptima.
- Los algoritmos metaheurísticos van desde simples algoritmos de búsqueda local hasta sofisticados procedimientos de aprendizaje.
- Son algoritmos heurísticos y normalmente no deterministas.
- Introducen mecanismos para evitar caer en subóptimos del espacio de búsqueda.
- Las metaheurísticas no son algoritmos específicos para un problema determinado.
- Pueden incluir conocimiento específico del problema que se pretende resolver –como las heurísticas–, pero este subproceso se controla mediante una estrategia de más alto nivel.
- Las metaheurísticas más modernas utilizan de manera implícita o explícita mecanismos de memoria para aprender durante el proceso de búsqueda.

Ejemplos de meteheurísticas son la Optimización con Colonia de Hormigas (ACO)[16]– [94–96], los GAs[17] [153, 174, 310] y los Algoritmos Evolutivos (EAs)[18] [28, 29, 107] en general, la Búsqueda Local Iterativa (ILS)[19] [91, 229], el Recocido Simulado (SA)[20] [1, 64, 90] o la Búsqueda Tabú (TS)[21] [38, 150, 152].

[16] *Ant Colony Optimization*
[17] *Genetic Algorithms*
[18] *Evolutionary Algorithms*
[19] *Iterated Local Search*
[20] *Simulated Annealing*
[21] *Tabu Search*

Algunas de estas metaheurísticas se inspiraron en procesos naturales como la evolución, en cambio otras son extensiones sofisticadas de técnicas tales como los algoritmos voraces (por ejemplo, el Procedimiento Adaptativo de Búsqueda Voraz Aleatorizada (GRASP)[22] [112]) o la búsqueda local (por ejemplo, la Búsqueda Local Estocástica (SLS)[23] [177]). Durante los primeros tiempos de investigación en metaheurísticas, los investigadores trabajaban en las diferentes técnicas sin que hubiera mucha interacción entre ellas, debido fundamentalmente a que las metaheurísticas puras estaban teniendo un éxito considerable. Solo cuando empezó a estar claro que el purismo algorítmico estaba frenando el avance del campo la comunidad científica empezó a plantearse utilizar combinaciones de ellas en un mismo algoritmo. Esto dio lugar a la hibridación, es decir, al nacimiento de las metaheurísticas híbridas.

Lo que hace más interesante la hibridación de diferentes algoritmos es que se pueden explotar características complementarias de varias técnicas diferentes dando lugar a un algoritmo más robusto y potente, y por lo tanto, a una estrategia de optimización mejor. De hecho, seleccionar la combinación adecuada de técnicas para crear un algoritmo que resuelva un problema de optimización complejo puede ser determinante para lograr el funcionamiento y rendimiento deseados. A su vez, esta es una tarea considerablemente difícil, que requiere de un amplio conocimiento de todas las técnicas existentes en las distintas áreas de optimización, además de un conocimiento del problema sobre el que se quiere aplicar, ya que problemas diferentes pueden necesitar técnicas muy distintas o combinaciones de ellas muy dispares.

2.2.2. Clasificación

Una cuestión importante es la clasificación de las metaheurísticas. Existen múltiples formas para clasificar los algoritmos metaheurísticos, en función de las características que se seleccionen. Una de ellas es por el origen del algoritmo, es decir, si están inspirados en la naturaleza o no lo están. Ejemplos de algoritmos inspirados en la naturaleza son los EAs (que se estudiarán con más detalle en una sección posterior) que están inspirados en los procesos de la evolución natural de las especies y los algoritmos de enjambre[24] [45, 193] que están inspirados en el comportamiento social de diferentes especies animales como por ejemplo la ACO o la Optimización por Enjambre de Partículas (PSO)[25] [192]. Entre los algoritmos no inspirados en la naturaleza se puede citar la TS o la Búsqueda Dispersa (SS)[26] [152]. Las metaheurísticas también se pueden clasificar en función de cómo usan la función objetivo [45]. Hay algoritmos que definen una función objetivo al principio del algoritmo y esta se mantiene fija durante toda su ejecución (como en

[22] *Greedy Randomized Adaptive Search Procedure*
[23] *Stochastic Local Search*
[24] *Swarm Intelligence*
[25] *Particle Swarm Optimization*
[26] *Scatter Search*

la mayoría de metaheurísticas); en cambio otros pueden ir alterando esta función objetivo incorporando información que se va capturando durante el proceso de búsqueda (como en la Búsqueda Local Guiada (GLS)[27] [354, 355]). Otra posible clasificación puede ser en función del número de estructuras de vecindad que se pueden definir. La mayoría de metaheurísticas utilizan un único vecindario, pero se pueden definir varios, alterando por lo tanto la topología definida para el algoritmo de forma dinámica como por ejemplo con la Búsqueda en Vecindades Variables (VNS)[28] [252]. Esto permite diversificar la búsqueda en momentos en los que el algoritmo podría estar estabilizado en un óptimo local. Los algoritmos metaheurísticos pueden clasificarse también por el uso o no de información histórica, es decir, si son con memoria o sin memoria. Los algoritmos sin memoria funcionan como cadenas de Markov, es decir, la decisión sobre cuál es el próximo estado viene determinada únicamente por la información disponible en el estado actual. Por otro lado, los algoritmos con memoria utilizan toda la información que han ido recopilando durante el proceso de búsqueda para tomar decisiones sobre el futuro. El uso de la memoria puede realizarse tanto de forma explícita como en la TS o de forma implícita como en el caso de los EAs.

Existe una clasificación adicional a todas las anteriores que es la que suele utilizarse de forma más habitual. Se trata de una clasificación en función del número de soluciones mantenidas simultáneamente y empleadas en la toma de decisiones del algoritmo. Los algoritmos que únicamente tratan con una solución cada vez se denominan *métodos de trayectoria*. Por el contrario, los algoritmos que utilizan un conjunto de soluciones del espacio de búsqueda se denominan *metaheurísticas basadas en población*:

- *Basadas en trayectorias.* Funcionan con una única solución que se va desplazando por el espacio de búsqueda describiendo una trayectoria en él. El algoritmo comienza a partir de una solución inicial s_0 creada a partir de una función de inicialización ι que la genera aleatoriamente o mediante un procedimiento heurístico y produce una nueva solución en cada paso en función de la solución actual:

$$s_0 \leftarrow \iota() \tag{2.2}$$
$$s_{i+1} \leftarrow \sigma(s_i) \tag{2.3}$$

  Por ejemplo, si $\sigma \triangleq \arg\max \circ \mathbf{map}\ f \circ \mathcal{N}$ nos encontramos ante un Algoritmo de Escalada (HC)[29] en su variante de la pendiente más fuerte. El algoritmo puede o no disponer de memoria (puede o no conocer las decisiones tomadas en el pasado). Algunos ejemplos son el SA, la TS o la ILS.

- *Basadas en poblaciones.* Utilizan un conjunto de soluciones (población) que van evolucionando hasta que finaliza el algoritmo. Cuando ha finalizado se

[27] *Guided Local Search*
[28] *Variable Neighborhood Search*
[29] *Hill Climbing*

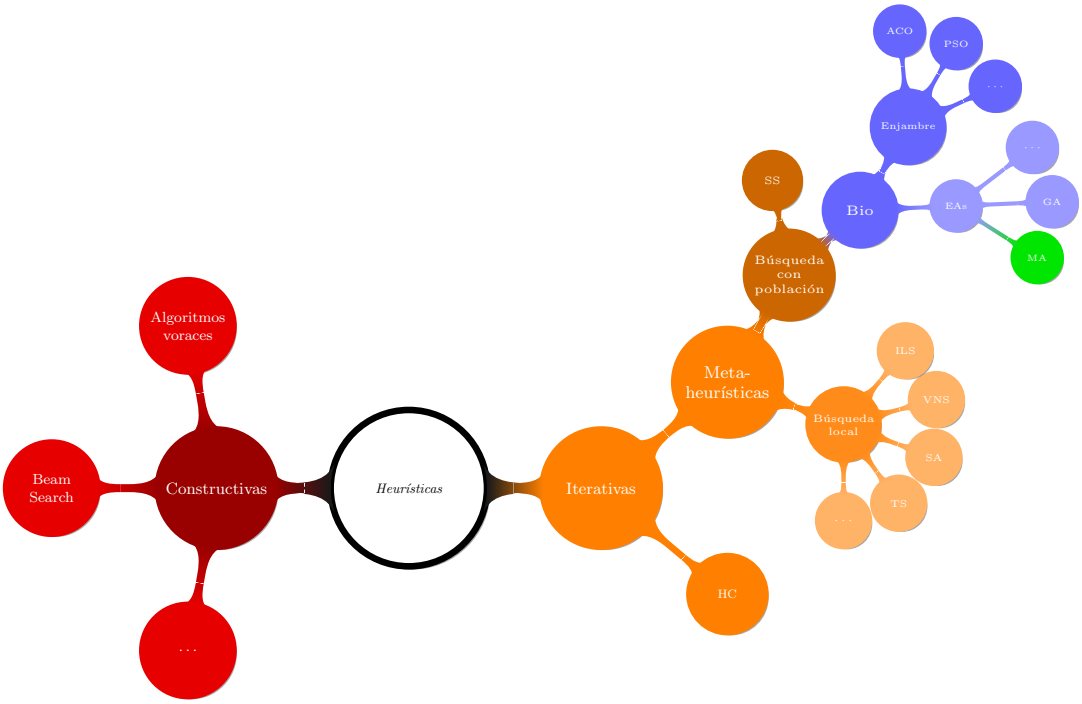


Figura 2.2: Clasificación de las técnicas heurísticas.

puede seleccionar la mejor de las soluciones encontradas de entre toda la población (o mantener una memoria de la mejor solución generada). Estos métodos extienden los anteriores al basar la toma de decisiones para la creación de nuevas soluciones no en una única solución actual sino en un conjunto de ellas. Algunos ejemplos son los GAs, Algoritmos Meméticos (MAs)[30], ACO o PSO.

En la Figura 2.2 se muestra un resumen de la clasificación de las técnicas heurísticas con ejemplos concretos de algoritmos específicos de cada una de ellas en los nodos hoja. Las aplicaciones de las metaheurísticas son múltiples [151, 159, 283], tanto para la industria como para la ciencia, lo que ha propiciado grandes esfuerzos de los investigadores en este campo que han producido importantes avances en las últimas décadas. La siguiente sección se va a centrar en la computación evolutiva, por ser una de las técnicas inspiradas en la Naturaleza que se toma como referencia.

2.3. Computación Evolutiva

La adaptabilidad al entorno natural es una capacidad presente en los seres vivos y en la Naturaleza misma. La grandeza de esta circunstancia se acrecienta al tener en cuenta que los entornos en los que la vida debe progresar están en

[30] *Memetic Algorithms*

constante cambio. Sin embargo, prácticamente en cualquier entorno la vida se abre camino, desde los lugares más inhóspitos a los más placenteros; si bien es cierto que no todos los seres vivos logran adaptarse a los cambios constantes que se producen en los ecosistemas de que forman parte. Esto implica que los seres que logran adaptarse sobreviven a los cambios, evolucionando y transformándose de forma paralela al entorno que les rodea y generando así más vida, incluso en ocasiones, nueva vida. Los organismos vivos que, por el contrario, no dispongan de esta capacidad de adaptación, no sobrevivirán. Otro punto interesante es que un mecanismo supra-orgánico como la evolución natural es el que facilita que los seres vivos cambien de manera que puedan adaptarse a nuevos entornos, puesto que estos no ejercen un esfuerzo consciente para evolucionar.

La Computación Evolutiva (EC)[31] trata de aprender y usar para su propio beneficio las lecciones de la Naturaleza para aplicarlas a la resolución de diversos problemas de optimización. Por lo tanto, la computación evolutiva es un conjunto de técnicas de optimización cuyo funcionamiento está basado en procesos biológicos. Esta definición es muy amplia y trata de incluir las numerosas aproximaciones existentes en la actualidad en este campo [29].

La EC es un área de la AI que hace referencia a diversos procedimientos de resolución de problemas complejos que utilizan mecanismos extraídos de los procesos naturales de evolución [124]. Con anterioridad se habían desarrollado otros modelos como las Redes Neuronales Artificiales (ANNs)[32] para resolver problemas de aprendizaje artificial basándose en algún modelo biológico real (en este caso, las redes de neuronas de un ser vivo) o el SA para resolver problemas de optimización. La EC aporta principalmente un mecanismo de resolución de problemas basado en la selección de soluciones potenciales y la generación de nuevas soluciones por recombinación de las ya existentes, de forma similar a como sucede en la evolución real de las especies en la naturaleza. La EC intenta, por lo tanto, capturar ideas de la Teoría de la Evolución de las Especies, pero sin pretender seguir un proceso estricto de simulación.

La rapidez con la que progresa este campo, hace que haya un creciente conjunto de técnicas que pueden considerarse "evolutivas". Sin embargo, todas ellas tienen unas características comunes. La siguiente cita de Jones [188] ilustra bien los puntos en común:

> Los algoritmos mantienen un conjunto de soluciones potenciales para un problema considerado. Algunas de estas posibles soluciones se usan para crear nuevas soluciones potenciales mediante el uso de *operadores*. Los operadores se aplican y producen nuevas soluciones potenciales. Las soluciones sobre las que se aplica un operador se seleccionan en función de la calidad de estas para resolver el problema en cuestión. Estos algoritmos repiten este mismo proceso para generar nuevos conjuntos de soluciones potenciales hasta que se cumple un determinado

[31] *Evolutionary Computation*
[32] *Artificial Neural Networks*

criterio de finalización.

Esta definición sugiere que si esta técnica trata con un conjunto de soluciones a la vez, es lógico pensar que tiene que establecerse algún tipo de mecanismo para determinar cuáles de ellas son mejores, seguramente incluyendo también una relación de orden. La definición anterior en ocasiones se expresa con un lenguaje más técnico extraído de las ciencias biológicas, usando términos como *genes*, *cromosomas*, *población*, etc.

El origen de la EC se encuentra a finales de los años '60 en los trabajos que varios investigadores desarrollaron durante esta época. Se puede citar a algunos de ellos, por ejemplo Holland, que comienza a incorporar fenómenos tomados de la teoría de la evolución, como la selección natural o la superviviencia del más apto para resolver problemas computacionales. Sus investigaciones iniciales se plasmaron en el libro *Adaptation in Natural and Artificial Systems* [174] en el que plantea las bases de los GAs. Otros investigadores que contribuyeron en la misma época al auge de la EC con diversos estudios relacionados con la evolución y su simulación computacional son Fogel [126] (pionero de la programación evolutiva) y Rechenberg [306] y Schwefel [324] (creadores de las estrategias evolutivas). Para la mayoría de científicos de este campo este puede considerarse el inicio de la EC (pero véase también [124]). En sus comienzos, las ideas de estos y otros muchos científicos que se fueron uniendo a este campo eran estrictamente académicas, tratando de ampliar el horizonte de la AI hacia nuevas áreas, pero a partir de los años '80 la EC comienza a aplicarse a la resolución de problemas de múltiples ámbitos como la ingeniería, la medicina o la economía [85, 153].

2.3.1. Evolución Biológica

Los científicos han tratado de comprender y explicar el funcionamiento de la evolución de las especies a través de diversas teorías, pero no fue hasta el siglo XIX cuando el desarrollo de otras disciplinas como la anatomía comparativa permitió que se postularan los principios fundamentales que rigen el funcionamiento de la evolución [5]. Estos principios están basados en los trabajos de Darwin [83] sobre la selección natural y de Mendel [239] sobre la herencia genética. A continuación se detallan algunos de los aspectos más relevantes [246]:

- La evolución es un proceso que no opera directamente sobre organismos, sino sobre cromosomas.
- La estructura y las características de un ser vivo se codifican a través de los *cromosomas*.
- La información que contienen los cromosomas va pasando de una generación a la siguiente por medio de la reproducción.
- El proceso evolutivo tiene lugar durante la reproducción. La Naturaleza dispone de una miríada de estrategias reproductivas. Dos de ellas son la

mutación que introduce variabilidad en el conjunto de genes propios, y la *recombinación* que produce el intercambio de información genética entre individuos diferentes.

- La selección natural es un mecanismo que relaciona los cromosomas con la idoneidad de las entidades que representa, favoreciendo la eficacia y la adaptación al entorno y por el contrario, la extinción de los individuos menos eficaces y menos adaptados.

La Teoría Sintética [184], se basa en los principios enumerados, aunque no es la única, ya que existen otras como la Teoría Neutral [195] o la Teoría del Equilibrio Interrumpido [161] que proponen algunos matices, pero sí es la que se considera como el modelo básico. Por otro lado, es interesante significar que estos principios fundamentales tan básicos dotan a la Naturaleza de una enorme capacidad para el desarrollo y expansión de nuevas formas de vida. Por ello parece lógico que esta capacidad haya atraído en algún momento a los científicos e investigadores hacia la idea de aplicar estos conceptos a las Ciencias de la Computación –más concretamente a la algoritmia– para disponer de herramientas computacionales que sean capaces de poseer características similares. Un aspecto interesante de la evolución es que se trata de un proceso reactivo, es decir, los organismos cambian –se adaptan– como respuesta a los estímulos producidos por las variaciones de su propio entorno. Esto quiere decir que la evolución no tiene un objetivo predefinido, por el contrario, los sistemas evolutivos artificiales sí tienen un objetivo predeterminado de antemano, que además, es deseable alcanzar lo más rápido y eficientemente posible. Según esto, los sistemas bio-inspirados se pueden clasificar en función de dos criterios diferentes:

- Simular la Naturaleza, es decir, en este caso se trata de intentar reproducir los principios de la Naturaleza con el mayor rigor posible.
- Adaptar estos principios eficientemente a los requisitos del problema que se pretende resolver, pero sin que se cumplan rigurosamente los principios de la propia Naturaleza.

Estas son las dos líneas de investigación principales en la actualidad para los científicos en este campo. La primera tiene más importancia en el campo de la Vida Artificial [211] ya que permite recrear y estudiar numerosos fenómenos naturales, tales como el parasitismo, las relaciones entre los depredadores y sus presas y otros muchos. La segunda aproximación se puede considerar más práctica y constituye la fuente de ideas principales para los EAs. Evidentemente estas dos ideas no son herméticas, sino que frecuentemente se solapan produciendo también resultados de éxito.

2.3.2. Algoritmia en la Computación Evolutiva

Los EAs son procedimientos estocásticos de búsqueda basados en la teoría de la evolución. Están basados en el principio de supervivencia del más apto a través

de su adaptación al entorno, tal y como sucede en la Naturaleza. Evidentemente los individuos más aptos son los que tienen más posibilidades de supervivencia y por lo tanto, de transmitir su herencia genética a las generaciones posteriores. El punto de vista de los EAs es eminentemente práctico ya que intentan usar ideas de la evolución natural para resolver un cierto problema. El punto central de la discusión es ahora la optimización y ver cómo se puede lograr este objetivo. Hay numerosos estudios sobre este tipo de algoritmos demostrando que son una muy buena opción para abordar la resolución de problemas de optimización [28, 29, 45, 125, 328].

Más concretamente, los EAs son metaheurísticas de propósito general inspiradas en la evolución de las especies que utilizan un conjunto de operadores como la selección, la mutación y el cruce para hacer evolucionar una población de individuos. Cada individuo representa una solución candidata del problema considerado, la cual es sometida a diversas transformaciones y finalmente a un proceso de selección que favorece la permanencia en la población de las mejores soluciones. Este ciclo completo se repite un número determinado de generaciones –es decir, de iteraciones– o hasta que se cumpla una condición predefinida de antemano. El objetivo es que esta evolución artificial sea capaz de explorar amplias regiones del espacio de búsqueda usando uno de los principios fundamentales de la Teoría de la Evolución de las Especies, es decir, la supervivencia de los mejores, generando buenas soluciones para el problema que se esté considerando. Para que el proceso completo funcione correctamente hay que introducir mecanismos que dirijan el proceso de búsqueda –la selección– y mecanismos que permitan introducir variabilidad en las soluciones candidatas –recombinación o cruce. En definitiva, un EA debe disponer de:

- Una adecuada representación de las soluciones, es decir, el dominio del problema debe codificarse en las soluciones de una manera determinada que permita el tratamiento correspondiente.

- Un conjunto de soluciones candidatas que representen la población de individuos del problema.

- Una función de aptitud –también denominada función de fitness– que determine la calidad de las soluciones candidatas.

- Procedimientos de selección y reemplazo basados en la función de aptitud.

- Un mecanismo que transforme unas soluciones candidatas en otras nuevas, es decir, un procedimiento para construir nuevas soluciones a partir de las existentes.

La Figura 2.3 representa un esquema básico de funcionamiento de un EA. En los siguientes apartados se detalla con más profundidad lo visto hasta ahora.

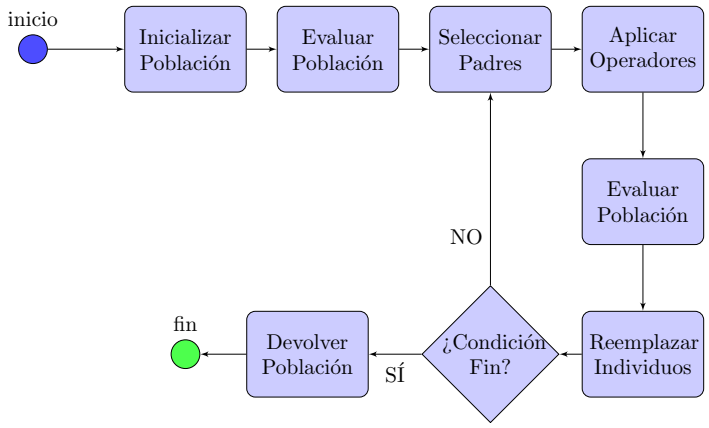


Figura 2.3: Diagrama de flujo con el funcionamiento de un EA.

2.3.2.1. Función Objetivo

El procedimiento de búsqueda de un EA está guiado fundamentalmente por una función objetivo –función de aptitud o fitness en adelante– característica del problema que se pretende resolver, por lo que este se puede considerar un componente esencial. Esta función proporciona una medida de la idoneidad para cada una de las soluciones candidatas que forman la población del algoritmo, es decir, una medida de su calidad. Habitualmente a la calidad de las soluciones se le asocia el término anglosajón *fitness*[33]. Por lo tanto, esta función de aptitud se define de forma que a cada solución se le asigna un valor real que mide la calidad de dicha solución, es decir:

$$f : S \to \mathbb{R} \tag{2.4}$$

Hay dos aspectos sumamente importantes a considerar para que la función de aptitud esté correctamente definida:

- *El criterio de codificación*: las soluciones candidatas deben codificarse de la manera adecuada para que puedan ser tratadas por la función de aptitud f, es decir, hay que especificar el procedimiento –codificación– que establece la correspondencia entre cada punto del dominio del problema –solución– con la representación de dicha solución en el EA.

- *El tratamiento de los individuos no factibles*: Hay problemas para los que no es posible establecer la correspondencia anterior entre cada punto del dominio del problema y el conjunto de las soluciones candidatas para resolverlo. La consecuencia de esto es que no todas las soluciones candidatas son elementos válidos del espacio de búsqueda del problema considerado. Es frecuente utilizar términos de penalización en este caso.

[33]La traducción literal de *fitness* es idoneidad o aptitud. De aquí en adelante se utilizará este termino para referirse al concepto de calidad de las soluciones

Una consideración adicional de la función de aptitud es que es muy importante desde un punto de vista estrictamente computacional, ya que el coste y por lo tanto la eficiencia de un EA se acostumbra a medirlo proporcionalmente al número de evaluaciones de las soluciones candidatas realizadas mediante esta función. El fitness en la mayoría de ocasiones se puede calcular mediante una ecuación matemática más o menos compleja, aunque en otras ocasiones puede ser necesario recurrir a sofisticadas simulaciones de sistemas físicos. También es necesario remarcar que la función de fitness puede incorporar algún tipo de ruido [166, 345] o variar dinámicamente a lo largo de la ejecución del EA. Dada la importancia de la función de fitness, todos los restantes componentes del EA deben tener en cuenta las características propias de esta función para el problema que se pretende resolver.

2.3.2.2. Inicialización y Parada

Cuando el EA comienza su ejecución debe disponer de una población de individuos iniciales. La inicialización consiste en establecer cómo se debe construir este conjunto de individuos que forman la población inicial del EA. Lo más habitual es que estos individuos se generen de forma aleatoria al comienzo del algoritmo. Dos cuestiones relevantes que hay que considerar son:

- El tamaño de la población: si el tamaño de la población –y por lo tanto, de la población inicial– es lo suficientemente pequeño se corre el riesgo de que la muestra inicial no sea lo suficientemente representativa y que no abarque el área adecuada del espacio de búsqueda.

- La cardinalidad del alfabeto utilizado para la codificación de las soluciones: si el número de símbolos para codificar las soluciones es reducido con un tamaño de población más bajo puede ser suficiente para tener una muestra lo suficientemente representativa del espacio de búsqueda, en caso contrario se corre el riesgo de tener la búsqueda demasiado orientada hacia una dirección disminuyendo la probabilidad de explorar regiones más amplias del dominio del problema, lo que se puede subsanar si se inicializa la población en diversas zonas del espacio de búsqueda.

Una solución para poder tratar con estas dos cuestiones puede consistir en disponer de procedimientos de inicialización guiados [308] que garanticen que todos los símbolos del alfabeto tienen una representación uniforme en el conjunto de soluciones iniciales. Otra alternativa es introducir mecanismos heurísticos en el proceso de inicialización lo que puede tener un efecto beneficioso en términos de convergencia y de calidad de las soluciones [79, 305]. Estas estrategias de inicialización guiadas pueden tener como contrapartida la convergencia prematura del algoritmo. En cualquier caso, una parametrización adecuada del EA puede ayudar a conciliar tanto que la inicialización alcance una zona amplia del espacio de búsqueda como evitar el estancamiento prematuro del EA.

Con respecto a la parada, hay que establecer las condiciones que determinen que el EA finalice, es decir, cuándo el EA ha encontrado una solución lo suficientemente válida o por el contrario, cuándo ha fracasado en la búsqueda y no tiene sentido prorrogar el proceso [148]. Esto se puede hacer de diversas formas:

- Por el número de generaciones, es decir cuando el algoritmo alcanza un número predeterminado de generaciones (iteraciones).
- Cuando se alcanza una determinada tasa de convergencia del conjunto de soluciones, por ejemplo, cuando la entropía de las soluciones tiene un valor lo suficientemente próximo a 0.
- Cuando el coste del algoritmo ha alcanzado un determinado valor –como se ha mencionado con anterioridad el coste se suele medir en función del número de evaluaciones de los individuos.
- Se pueden utilizar métodos de inferencia probabilística para determinar con cierto grado de certeza si el algoritmo se encuentra estancado [72, 183].

2.3.2.3. Selección

La selección es el mecanismo que dirige el proceso de búsqueda favoreciendo a los individuos más aptos. Consiste en muestrear la población de tamaño n obteniendo k individuos que actuarán como progenitores de la nueva generación. El criterio de elección suele ser un parámetro característico del EA, aunque los más habituales suelen ser:

- *Muestreo directo*: se elige un subconjunto de tamaño $k \leqslant n$ de la población, siguiendo un criterio determinado. Por ejemplo, se eligen los k mejores individuos de la población.
- *Muestreo aleatorio o por torneo*: se seleccionan de forma equiprobable m individuos de la población eligiendo al mejor de estos m y repitiendo el proceso completo k veces.
- *Muestreos estocásticos*: se asignan probabilidades de selección a los individuos de la población en función de su aptitud. La elección de los k individuos se realiza considerando estas probabilidades de selección.

2.3.2.4. Operadores de Variación

Los operadores de variación son los mecanismos que transforman las soluciones candidatas en nuevas soluciones, es decir, producen la nueva generación de individuos –descendencia– a partir de los individuos seleccionados durante el proceso de selección. En los EAs existen al menos dos tipos de operadores de variación: la recombinación y la mutación.

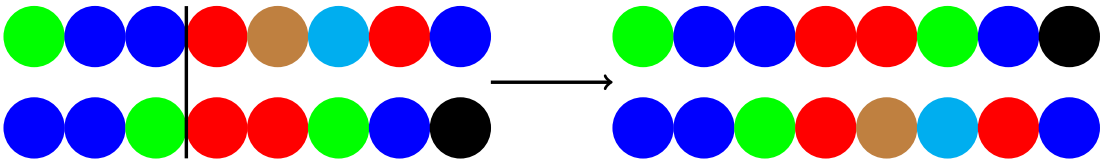


Figura 2.4: Ejemplo de operación de cruce para dos individuos.

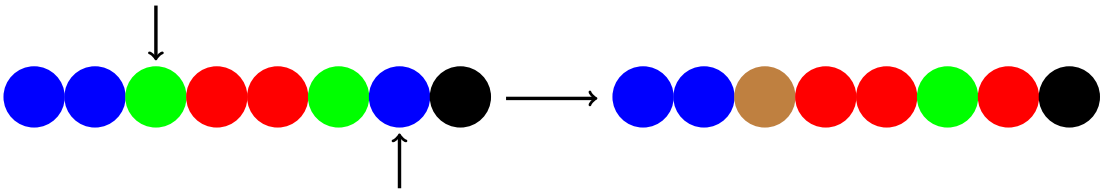


Figura 2.5: Ejemplo de operación de mutación.

Los operadores de recombinación –representados a través del *cruce*– actúan sobre grupos (típicamente parejas) de individuos originando normalmente otro conjunto de individuos que combinan las características de sus progenitores. En la Figura 2.4 se representa un esquema de operación de cruce entre dos individuos. Como se puede observar, cada individuo está formado por ocho genes y la recombinación tiene lugar a partir del cuarto gen, es decir, los genes del cuarto al octavo se intercambian entre los dos individuos originales produciendo dos nuevos individuos que tienen los tres primeros genes de uno de sus progenitores y los cinco últimos del otro. El ejemplo mostrado corresponde a cruce de un punto, pero existen otros muchos esquemas posibles (cruce de 2 puntos, uniforme, etc.), en particular cuando las soluciones tienen una estructura algebraica no trivial (permutaciones, conjuntos de k elementos, etc.).

Los operadores de mutación, en cambio, modifican directamente la información genética codificada en un individuo concreto, es decir, actúan sobre un único individuo. En la Figura 2.5 se representa una operación de mutación. En este caso, la mutación actúa sobre los genes tres y siete, alterando el valor de estos genes. Al igual que en el caso del cruce, la mutación puede realizarse de maneras muy diversas en función del problema considerado.

En los EAs el proceso de búsqueda se realiza durante esta fase de reproducción, siendo los operadores de recombinación y de mutación responsables de diferentes facetas del proceso. Los primeros realizan principalmente la búsqueda explotando las características propias de los individuos de la población actual. Por el contrario, la mutación actúa sobre la búsqueda explorando nuevas regiones del espacio de búsqueda intentado conseguir mejores soluciones.

2.3.2.5. Reemplazo

El reemplazo es el procedimiento por el que se sustituyen individuos existentes en la población por otros nuevos, generados a partir del proceso de selección y

aplicación de los operadores evolutivos. El proceso consiste en obtener, a partir de los n individuos que forman parte de la población y de los k descendientes, una nueva población de tamaño n. Para realizar el proceso de elección de n individuos del total de $n + k$ individuos existen varios criterios:

- *Reemplazo inmediato*: los k descendientes sustituyen a sus respectivos progenitores.
- *Reemplazo por parecido*: los k descendientes sustituyen a aquellos individuos miembros de la población que más se les parezcan.
- *Reemplazo de los peores*: se seleccionan los k peores individuos de la población y se sustitiyen por los k descendientes.
- *Reemplazo aleatorio*: se seleccionan de manera aleatoria k individuos de la población y son estos los que se sustituyen por los k descendientes.
- *Reemplazo completo*: la población de descendientes tiene tamaño $k = n$, es decir, se genera una población de descendientes del mismo tamaño que la población original, y se sustituye toda la población por la descendencia.
- *Reemplazo por inclusión*: se construye una nueva población de tamaño $n+k$ formada por los n individuos de la población original y los k descendientes y se muestrean de la población total n individuos (normalmente los mejores).

2.3.3. Variantes de los Algoritmos Evolutivos

La discusión anterior, aunque de ámbito general, se adapta perfectamente a los GAs en particular [153, 174], el modelo concebido originalmente por Holland y en el que una de las propiedades más distintivas es el papel central que juega la recombinación durante el proceso de búsqueda. Existen en cualquier caso otros modelos de EAs, algunos de los cuales se enumeran a continuación:

- *Programación Evolutiva (EP)*[34]: se originó en el trabajo de Fogel *et al.* [126] y a diferencia de otras técnicas se focaliza más en la adaptación intrínseca de los individuos mismos y de su comportamiento que en la evolución de la información genética. Los individuos en las EP se codifican de forma compleja mediante autómatas finitos o estructuras algebraicas como los grafos. Una cuestión que merece la pena recalcar es que esta técnica utiliza exclusivamente la mutación como operador de variación.
- *Estrategia Evolutiva (ES)*[35]: fue desarrollada inicialmente por Rechenberg y Schwefel [306, 324] y se diseñó con el objetivo de resolver problemas de

[34] *Evolutionary Programming*
[35] *Evolution Strategy*

ingeniería. Tiene la particularidad de que la representación de cada individuo esta formada por 2 conjuntos de variables reales: por un lado están las variables que representan el espacio de búsqueda y por otro un conjunto de parámetros usados para controlar la operación de mutación (típicamente realizada mediante perturbación gaussiana). Al estar representado cada individuo por dos conjuntos diferentes de valores, los operadores de variación hay que diseñarlos de forma que su comportamiento sea el adecuado para cada uno de ellos, es decir, el mismo operador puede funcionar de una forma sobre las variables que representan la solución y de otra diferente sobre los parámetros de mutación.

- *Programación Genética (GP)*[36]: es una extensión de los algoritmos genéticos basada en los trabajos de Cramer y Koza [81, 198] que tiene por objetivo evolucionar programas. Típicamente, aunque no siempre, se representa mediante estructuras arbóreas, lo que implica la adaptación de todo el aparataje de operadores a este contexto.

- *Evolución Diferencial (DE)*[37]: es una técnica para resolver problemas continuos originalmente ideada por Storn y Price [339] para resolver el problema del ajuste polinómico de Tchebychev. Comparte características propias de la EC como el operador de cruce y de la PSO como el reemplazo uno a uno. En la DE la población inicial se genera de forma pseudo-aleatoria y en cada generación cada individuo realiza una mutación (a partir de 3 individuos diferentes extraídos aleatoriamente de la población), cruzándose finalmente este último con el individuo original y reemplazando al original de la población en caso de tratarse de una solución de mejor calidad.

- *Algoritmos de Estimación de Distribuciones (EDAs)*[38]: fueron desarrollados originalmente por Mühlenbein y Paaß [259] y a diferencia de los anteriores en vez de intentar encontrar la solución óptima directamente buscan estimar una distribución de probabilidad de la misma. Esta distribución de probabilidad se actualiza en cada generación a partir de los mejores individuos de la generación previa (obtenidos a su vez mediante el muestreo de la distribución de probabilidad anterior). Un aspecto importante a considerar es que no existen operadores de variación, sino únicamente muestreo y actualización del modelo probabilístico.

2.4. Computación Memética

Los EAs son técnicas de optimización que funcionan frecuentemente sin conocimiento específico del problema (excepto por la función de fitness), hecho con-

[36] *Genetic Programming*
[37] *Differential Evolution*
[38] *Estimation of Distribution Algorithms*

siderado incluso como una ventaja [153]. Sin embargo, la necesidad de explotar este conocimiento se ha repetido numerosas veces en la historia reciente de estas técnicas, debido fundamentalmente a la idea de que con ello se pueden mejorar los resultados tanto en términos de eficiencia como de la calidad de las soluciones encontradas [85, 169]. Los MAs [170, 255, 256, 258, 263] son posiblemente uno de los exponentes más significativos de estas técnicas que combinan las estrategias basadas en población con la incorporación de conocimiento específico del problema, habiéndose demostrado que son muy válidos en un amplio número de problemas y más concretamente en la aproximación de soluciones para problemas **NP**-duros. A diferencia de los EAs, los MAs están concebidos para explotar todo este conocimiento disponible del dominio del problema, siendo esta una característica primordial de su diseño y no un mero mecanismo opcional [256, 257].

2.4.1. Antecedentes y Necesidad

La característica distintiva de los MAs mencionada con anterioridad –la inclusión de conocimiento del problema en el proceso de búsqueda– está también fundamentada en resultados teóricos. Como establecieron Hart y Belew [169] inicialmente y Wolpert y Macready [368] popularizaron posteriormente mediante el Teorema de No Free Lunch (NFLT)[39], los algoritmos de búsqueda tienen un rendimiento estrictamente en consonancia con la cantidad y calidad de conocimiento del problema del que disponen.

El NFLT, que es uno de los teoremas fundamentales relacionados con la búsqueda y la optimización, propone que si consideramos todos los algoritmos de optimización como una caja negra[40], en este caso todos ellos serían equivalentes en promedio [367, 368]. De manera más precisa:

Definición 21. *Sea A un algoritmo, $f : \mathcal{S} \to \mathcal{R}$ una función objetivo (donde $\mathcal{S}$ y $\mathcal{R}$ son finitos) y sea $m \in \mathbb{N}$. Se define $d_A^{f,m} : \mathcal{S}^m \to \mathbb{R}$ como una distribución de probabilidad sobre secuencias de m elementos distintos del espacio de búsqueda $\mathcal{S}$ visitados por A cuando se optimiza la función f y se permiten m invocaciones de la misma, es decir, la probabilidad de que A genere dicha secuencia.*

Según la definición anterior, si A es un algoritmo determinista entonces $d_A^{f,m}(\vec{s}) = 1$ para una cierta secuencia $\vec{s} \in \mathcal{S}^m$ de puntos del espacio de búsqueda y $d_A^{f,m}(\vec{s}) = 0$ para cualquier otra secuencia. Si A no es determinista en general habrá múltiples $\vec{s}$ para los que $d_A^{f,m}(\vec{s}) > 0$.

Teorema 1. *Sean A_1 y A_2 dos algoritmos, entonces:*

$$\sum_{f \in \mathcal{F}} d_{A_1}^{f,m}(\vec{s}) = \sum_{f \in \mathcal{F}} d_{A_2}^{f,m}(\vec{s}) \tag{2.5}$$

[39] *No-Free-Lunch Theorem*

[40] se denomina caja negra a aquel elemento que es estudiado desde el punto de vista de las entradas que recibe y las salidas o respuestas que produce, sin tener en cuenta su funcionamiento interno

donde $d_A^{f,m}(\vec{s})$ *es la probabilidad de que el algoritmo A visite una secuencia* $\vec{s}$ *determinada de m puntos del espacio de búsqueda tras m ejecuciones de la función f y* $\mathcal{F}$ *es el conjunto de todas las funciones posibles entre* $\mathcal{S}$ *y* $\mathcal{R}$*.*

El teorema se aplica a todo tipo de algoritmos de optimización, tanto los deterministas como los estocásticos e incluso a los que son capaces de aprender y ajustar sus estrategias de búsqueda sobre la marcha. Asimismo es independiente de la forma en la que se mida el rendimiento de los algoritmos o del método de representación utilizado.

En resumen, según el NFLT, todas las técnicas de búsqueda existentes tienen un rendimiento equivalente, en promedio, en el conjunto de todos los problemas de optimización combinatoria, es decir, aunque unas se comporten mejor para un problema determinado y otras técnicas lo hagan mejor para otro problema diferente, en el conjunto total el promedio del rendimiento de todas las técnicas de búsqueda debe ser el mismo.

Cuando se publicó este teorema causó un notable pesimismo y confusión entre la comunidad científica, fundamentalmente en lo concerniente a la evaluación y comparación de las diversas metaheurísticas y algoritmos de inteligencia computacional existentes. La principal implicación del teorema es que la búsqueda del "mejor" algoritmo de propósito general de optimización o búsqueda no tiene sentido, ya que desde un punto de vista teórico no existe tal algoritmo. Hecho que es fácil de constatar siguiendo un razonamiento basado en un argumento del adversario [82].

Sin embargo, esto sigue generando cierta controversia, ya que existe una sensación generalizada sobre que los EAs en general son superiores a otras técnicas de optimización, fundamentalmente cuando el conocimiento sobre el problema considerado es limitado, lo que contradice a Wolpert y Macready con su NFLT. Droste et al. propusieron algunos matices en 1999 [104] argumentando que el escenario en el que el NFLT está basado no modela los problemas de optimización de la vida real y que cuando se tratan este tipo de escenarios sí existen diferencias en cuanto a eficiencia entre unas técnicas de optimización y otras. Además, Igel y Toussaint [185] sostienen que en la práctica no es necesario considerar todas las funciones de optimización posibles (problemas) y que si restringimos este conjunto entonces también vuelven a aparecer diferencias entre las diferentes técnicas de optimización. Por ejemplo, se ha demostrado que para las funciones pseudobooleanas, si se introducen restricciones de complejidad y se consideran subconjuntos de ellas entonces existen diferencias en el rendimiento entre las diversas técnicas conocidas (en [362] la complejidad se define en términos del número de mínimos locales y en [103, 104] la complejidad se define en términos del tamaño del diagrama de decisión binario ordenado más pequeño que representa la función). No obstante, también es cierto que se ha propuesto una nueva versión del NFLT que establece que los resultados del NFLT son válidos para cualquier subconjunto F de todas las funciones existentes si y solo si F es un conjunto cerrado bajo permutación (c.u.p. *closed under permutation*) [323]. Según este resultado es posible

inferir clases de funciones donde el NFLT no se sostiene porque estas clases no son c.u.p., pero también que hay clases reducidas de funciones que siguen sujetas al NFLT.

Una de las consecuencias de los resultados anteriores es la necesidad de emplear conocimiento del problema para garantizar un buen rendimiento del algoritmo de optimización. El término "hibridación" se suele utilizar para denotar aquellos procesos que incorporan conocimiento del problema para su resolución [85]. En su sentido más genérico, por hibridación se entiende la unión o mezcla de dos o más características heterogéneas, bien sea de manera deliberada o por la propia evolución natural. Cuando el proceso se dirige de forma consciente, lo habitual es que el objetivo sea conseguir una nueva característica mejorada. Uno de los primeros algoritmos considerado como memético se puede encontrar en 1988 [281] y consistió en la hibridación de un GA tradicional y un SA para resolver el TSP. Esta aproximación tuvo varias características que anticiparon muchas de las prácticas que todavía hoy en día se siguen utilizando, como por ejemplo la fase competitiva del algoritmo y la fase cooperativa.

2.4.2. Algoritmos Meméticos

Según la teoría filosófica de Dawkins la cultura humana se puede descomponer en unidades atómicas denominadas "memes". Cada uno de estos memes es una pieza básica de conocimiento que se puede modificar o combinar con otros memes e incorporar al cerebro humano. El adjetivo "memético" procede del término meme que fue acuñado por Dawkins en su libro "The Selfish Gene" [86] y denota una similitud con gen, pero en el contexto de una evolución cultural en vez de biológica o dicho de otra forma, un meme puede considerarse como la unidad de imitación o de transmisión de información cultural. Blackmore en su libro "The Meme Machine" [43] discute sobre el papel de los genes, los memes y las relaciones entre ellos. Dentro de una comunidad los memes menos interesantes acaban desapareciendo con el tiempo, en cambio, aquellos que sí lo son pueden acabar propagándose rápidamente impregnando toda la comunidad. La siguiente cita de Dawkins [86] puede ilustrar el concepto de meme:

> Ejemplos de memes pueden ser las canciones, ideas, lemas, la ropa de moda, la forma de hacer una vasija o de construir un arco. Al igual que los genes se propagan por sí mismos dentro del conjunto de genes saltando de cuerpo en cuerpo a través del esperma o de óvulos, los memes también se propagan por sí mismos entre el resto de memes saltando de cerebro en cerebro a través de un proceso que se puede denominar imitación, en su sentido más amplio.

Esta caracterización de un meme implica que en los procesos de evolución cultural, la información no se transmite simplemente entre individuos sin alterarse, sino que es procesada y mejorada mediante los mecanismos de comunicación entre los individuos transmisor y receptor. Estas ideas en cierta forma coinciden

con la teoría del modelo evolutivo de Lamarck de 1809 [210], el cual propone dos ideas fundamentales, una de ellas es que los caracteres se adquieren durante la vida y la otra es que las especies "transmutan" cada cierto tiempo. Por otro lado, muchos investigadores tienen la idea de que los memes únicamente se materializan como piezas de información restringida al cerebro, en cambio otros van más allá y amplían este horizonte al comportamiento. Esta metáfora de la evolución cultural puede trasladarse al campo de la optimización si se interpretan los memes como procedimientos de aprendizaje individual que utilizan los MAs durante el proceso de búsqueda para profundizar en las características propias del individuo durante la vida de este, de manera que la información transmitida por un individuo no es idéntica a la información adquirida por él (a diferencia de los genes). Este proceso de mejora se realiza en los MAs incorporando heurísticas, algoritmos de aproximación, técnicas de búsqueda local, operadores de recombinación especializados o métodos exactos, por citar solo algunas formas de hacerlo [256]. La mayoría de los MAs se pueden interpretar como estrategias de búsqueda en las que una población de agentes de optimización cooperan y compiten [281]. Desde un punto de vista algorítmico, cuando dos o más estrategias se combinan con el objetivo de conseguir ciertas sinergias, se puede mejorar la capacidad del algoritmo para resolver un problema determinado. El éxito de los MAs puede seguramente explicarse como consecuencia de esto.

Los MAs –al igual que los EAs– son metaheurísticas basadas en población, pero con la diferencia de que los individuos (en este caso denominados habitualmente *agentes*) son entidades activas que compiten y cooperan para mejorar la búsqueda de soluciones. Desde un punto de vista histórico, es conveniente conocer que cuando se concibieron los MAs en 1988, los GAs eran tremendamente populares en la comunidad científica debido fundamentalmente a que parecía que tenían un rendimiento superior a las demás técnicas de búsqueda conocidas hasta la fecha. Sin embargo, la posterior aparición del NFLT cambió el curso de la historia y empezó a comprenderse que el éxito no dependía tanto del algoritmo en sí como del conocimiento adicional que se tuviera sobre el problema considerado.

Existen numerosas formas de implementar un MA, de hecho, a diferencia de otros algoritmos no existe una plantilla general para ellos, ya que no se les puede considerar como simples algoritmos de optimización, sino más bien como un concepto nuevo que combina algoritmos de búsqueda local y global. Esto los hace sumamente potentes y flexibles y da lugar a un nuevo paradigma de optimización. Quizá la forma más habitual de implementarlos sea realizar una búsqueda local después de la evaluación. Esto convierte a los MAs en EAs con una búsqueda local incorporada. Según Moscato [254]:

> Mientras los GAs están inspirados en la emulación de la evolución biológica, los MAs intentan mimetizar la evolución cultural. La evolución memética es una unión entre la búsqueda global basada en poblaciones y la búsqueda local heurística realizada por cada uno de los individuos.

Algoritmo 1: Pseudocódigo de un algoritmo memético clásico

```
function MA (↓ I, ↓ par) returns S
// I: Instancia del problema
// par: Parámetros del problema
// S: Solución devuelta
pop ← InicializarPoblación(I, par);
pop ← EvaluarPoblación(I, par, pop);
while ¬ CondiciónFin do
    // Fase de Competición
    pop' ← SeleccionarPadres(I, par, pop);
    // Fase de Cooperación
    for i ← 1 to par.numop do
        pop' ← AplicarOperador(I, par, pop', par.Op[i]);
    end
    // Fase de Aprendizaje Individual
    pop' ← Mutación(I, par, pop');
    pop' ← MejoraLocal(I, par, pop');
    pop' ← EvaluarPoblación(I, par, pop');
    // Fase de Competición (continúa durante la selección)
    pop ← ActualizarPoblación(I, par, pop, pop');
    if Convergencia(pop) then
        pop ← RefrescarPoblación(I, par, pop);
    end
end
return Mejor(pop)
```

Un MA puede verse como un algoritmo híbrido que sigue un esquema evolutivo clásico con una etapa adicional de evolución cultural. El Algoritmo 1 describe su funcionamiento más básico. En él se puede comprobar que el MA tiene las mismas fases que un EA, pero con una operación adicional de aprendizaje individual que consiste en la aplicación de mecanismos de mejora local para aquellos individuos seleccionados previamente.

Como se ha mencionado con anterioridad, los MAs tienen tres fases fundamentales, cooperación, aprendizaje individual y competición, aunque antes de ellas está la inicialización de la población, es decir, la creación del conjunto inicial de soluciones. Los EAs en general realizan esta inicialización de forma aleatoria (en algunos casos se puede introducir alguna restricción para que se cubra una región lo suficientemente extensa del espacio de búsqueda). Sin embargo, los MAs intentan producir soluciones de calidad desde el principio, por lo que durante el proceso de inicialización intentan inyectar tales soluciones en la población, bien mediante algún mecanismo heurístico que permita incluir estas soluciones más prometedoras o utilizando la búsqueda local ya en esta etapa inicial. Con respec-

to a la fase de cooperación, esta se corresponde habitualmente con la aplicación de los operadores de recombinación (considérese que el número de operadores de recombinación no está limitado y podrían enlazarse en cascada cuantos operadores se desee. En el Algoritmo 1 hay *par.numop* operadores). Durante este fase los agentes cooperan entre sí intentando producir mejores soluciones. La segunda es la de aprendizaje individual, que incluye tanto la mutación como la operación de mejora local. La mejora local modela el aprendizaje individual[41] y puede ser simplemente una etapa de búsqueda local adicional para los individuos seleccionados. Finalmente está la fase de competición, en la que se reconstruye la población seleccionando aquellos agentes más prometedores y desechando los demás (también se incluye en esta fase el proceso de selección de individuos).

Según Eiben y Smith [106], existen ciertos lugares preferentemente en los que podría incorporarse conocimiento específico del problema en un MA. Algunas posibilidades son estas:

- Durante la inicialización de la población: normalmente los EA inicializan la población de forma aleatoria, en cambio los MAs pueden utilizar mecanismos para generar soluciones iniciales de alta calidad, bien a través de métodos heurísticos diseñados específicamente para ello o realizando una búsqueda local a partir de las soluciones generadas aleatoriamente en una primera ronda. Esto debe hacerse con cierto cuidado, ya que una población demasiado homogénea puede conducir al MA hacia una convergencia prematura.

- Durante la aplicación de los operadores: los operadores de mutación y recombinación se pueden diseñar de forma inteligente a partir del conocimiento del problema [65, 70, 78, 245]. De la misma forma también se pueden diseñar operadores de selección guiados por el conocimiento del problema [282].

- Incorporando conocimiento del problema en el proceso de obtención de fenotipos a partir de genotipos[42] [73, 77, 199, 234, 301, 309] creando una codificación entre cada miembro de la población y alguna característica del MA, por ejemplo, las probabilidades de cruce o mutación.

- En cualquier etapa intermedia se puede realizar una búsqueda local a partir de los individuos seleccionados.

El Algoritmo 1 incluye un componente adicional denominado RefrescarPoblación que tiene una importancia considerable, ya que afecta a la utilización eficiente de los recursos computacionales. Su funcionamiento consiste básicamente en detectar si la población ha convergido hacia individuos similares y por lo tanto, no hay suficiente variedad genética/memética para que el proceso de búsqueda

[41] *lifetime learning*
[42] *genotype-to-phenotype mapping*

pueda continuar avanzando. Una forma de detectar este escenario es a través de la medida de la entropía de Shannon [84], considerando un umbral mínimo a partir del cual se puede considerar esta convergencia como un hecho. Cuando eso sucede existen varias posibilidades, desde medidas pasivas como finalizar el algoritmo (se podría incluir este hecho en la condición de terminación del bucle correspondiente al ciclo de vida) u otras activas como puede ser reinicializar la población (o parte de ella) para introducir nueva variedad genética/memética.

2.4.3. Factores de Diseño de un MA

Según lo visto en la sección anterior, un MA puede considerarse un algoritmo compuesto por varias partes (inicialización, cooperación, mejora y competición). Esto hace de los MAs algoritmos lo suficientemente complejos como para que el diseño de los mismos sea decisivo en su rendimiento. El modelo básico explicado en la sección anterior se debe instanciar con un conjunto de parámetros adecuado para el problema que se pretenda resolver si se quiere obtener una verdadera herramienta potente de optimización. Las principales decisiones de diseño desde el punto de vista específico de los MAs tienen que ver con el componente de búsqueda local y no solo desde el punto de vista de su parametrización, sino también con respecto a su funcionamiento interno y la relación entre este y el resto de operadores de variación del algoritmo [343]. Por lo tanto, la primera consideración que se debe tener en cuenta debe ser la relación entre el componente de búsqueda local y los restantes operadores. Un parámetro fundamental y característico de los MAs es la frecuencia e intensidad del aprendizaje del individuo a través de la relación entre la búsqueda local (explotación) y la búsqueda global (exploración). Si la intensidad del aprendizaje de cada individuo –búsqueda local– es mayor se incrementa la probabilidad de que el algoritmo alcance un óptimo local pero puede incurrir en un coste computacional más elevado. Un buen ejemplo de esta cuestión se puede encontrar en el trabajo de Freisleben y Merz sobre el TSP [128], en el que consideran un procedimiento de búsqueda local intensivo –la heurística Lin-Kernighan [224]– y concluyen que la distancia media entre óptimos locales es similar a la distancia media entre un óptimo local y el óptimo global. Por esta razón introducen un operador de cruce que mantiene la distancia (DPX) generando hijos cuya distancia a sus padres es la misma que la distancia entre sus propios padres entre sí.

Una vez decidido el procedimiento de búsqueda local concreto que se va a utilizar hay que elegir la parametrización adecuada, es decir, determinar cuestiones como cuántas veces debe aplicarse –es decir, cada cuántas generaciones–, sobre qué individuos debe aplicarse –no tiene por qué aplicarse sobre todos los individuos, sino sobre una selección de ellos– y cómo debe ser la intensidad de la búsqueda –por ejemplo, se puede decidir acotar la profundidad de la búsqueda. Con respecto a las dos primeras cuestiones, las dos estrategias habituales son, o usar una frecuencia fija para decidir cuándo se ejecuta la búsqueda local y un valor fijo para el número de individuos sobre los que se aplica o bien tomar

la decisión en función de una distribución de probabilidad. Todas estas cuestiones son decisivas para lograr un algoritmo eficiente, de hecho, existen evidencias teóricas claras sobre el efecto que puede tener una parametrización inadecuada en el rendimiento del MA [208, 341–343] –un problema puede incluso pasar de ser fácilmente resoluble a serlo en tiempo no polinómico.

Como se ha mencionado previamente existen otros muchos parámetros que considerar, por ejemplo, decidir cuántos individuos van a recurrir a la búsqueda local. Si este número es elevado el coste computacional del algoritmo también se incrementa (evidentemente conforme este parámetro se aproxima a cero más próximo estará el MA de un EA). El resto de factores se pueden considerar similares a los del diseño de un EA, tal es el caso de decisiones sobre el tamaño de la población, el número de descendientes o las probabilidades de aplicación de los operadores de variación. A continuación se resumen algunos de ellos:

- La frecuencia del aprendizaje: cada cuánto tiempo se realiza la fase de aprendizaje individual o búsqueda local.
- La probabilidad de aprendizaje: qué porcentaje de los individuos van a disponer de la fase de aprendizaje individual o a qué subconjunto de soluciones de la población se le va a aplicar esta fase. Existe evidencia de que en ocasiones un aprendizaje Lamarckiano parcial puede ser beneficioso [71, 181].
- La intensidad del aprendizaje: durante cuánto tiempo o qué complejidad debe tener la fase de aprendizaje individual [343].
- El coste de la computación: un factor importante puede ser lidiar con el sobrecoste computacional que puede requerir la introducción de aprendizaje individual en el algoritmo completo, sobre todo cuando se trata de resolver problemas complejos del mundo real cuyas funciones de fitness tienen una complejidad elevada y no se pueden descomponer, a diferencia de lo que ocurre en muchos COPs [75]. Es frecuente usar en este tipo de problemas complejos modelos subrogados para acelerar la evolución [374].
- El mecanismo del aprendizaje: la forma en la que se pretende diseñar el mecanismo de aprendizaje local puede ser decisiva para conseguir un algoritmo con un alto nivel de eficacia (consecución de una solución de buena calidad) y de eficiencia (conseguir el mejor rendimiento computacional).

Relacionado con todo lo anterior, nótese que la fase de mejora local puede interpretarse en un MA como el resultado de la acción de un meme. En la siguiente subsección se va a abordar con mayor profundidad qué son los memes y cómo se pueden representar dentro de un MA.

2.4.4. Hacia la Computación con Memes

En la Sección 2.4.2 se vio el concepto de meme como elemento de evolución cultural y se estableció su relación con los MAs. Si se interpreta un meme como

un procedimiento para el aprendizaje de soluciones, un MA clásico trabajaría con memes implícitos dados por la selección del proceso de mejora local. Sin embargo, en la literatura se han estudiado otras alternativas diferentes sobre la forma en que los memes pueden incorporarse a los individuos, como la gestión explícita de los memes [200] (por ejemplo en los MMAs). En este caso, cada solución incluye los memes que determinan la forma en la que se realiza la mejora local. Estos memes evolucionan por sí mismos o junto a la solución completa y suponen un proceso de auto-adaptación como se verá en la sección siguiente.

La codificación de los memes tiene una ventaja importante y es que permite que estos evolucionen de forma dinámica durante la ejecución del MA. También es sabido que la AI ha tratado desde su origen de encontrar las fórmulas que permitieran diseñar estructuras que fueran no solo inteligentes sino también autónomas. Dentro del campo de optimización en inteligencia computacional, el objetivo último es ser capaces de detectar dinámicamente el algoritmo de optimización óptimo para resolver un problema determinado, esto es, el diseño automático de algoritmos de optimización eficientes en tiempo de ejecución. Desde este punto de vista, la Computación Memética (MC) se puede ver como la parte de la EC que se ocupa de estudiar estructuras más complejas formadas a partir de piezas básicas –memes– que interactúan entre sí para adaptarse dinámicamente con el objetivo de resolver el problema considerado. La MC se define, según Ong en [285], como:

> ...un paradigma que usa la noción de meme(s) como unidad de información codificada mediante representaciones computacionales con el propósito de resolver un problema.

En esta línea, según Neri y Cotta [262] una definición más moderna de MC puede ser:

Definición 22. *La MC es una disciplina amplia que estudia estructuras de computación complejas y dinámicas compuestas por módulos interactivos (memes) cuya dinámica de evolución está inspirada en la difusión de ideas. Los memes son simples estrategias cuya coordinación armónica permite la resolución de diversos problemas.*

Si se profundiza un poco más en la fase de aprendizaje individual, que recordemos es aquella en la que están más directamente involucrados los memes, esta se puede implementar de diversas formas. Una posibilidad es que el material memético se pueda incluir como parte del genotipo de las soluciones, por lo tanto, el meme decodificado de cada respectivo individuo se utilizaría para realizar la búsqueda local y a continuación se transmitiría el material memético de padres a hijos siguiendo un mecanismo de herencia. Otra posible opción es que los memes, por ejemplo, se puedan representar como patrones de reglas que capturen regiones del dominio del problema para aplicarlas en la fase de aprendizaje individual con el objetivo de encontrar soluciones más prometedoras.

Un aspecto importante es determinar cómo los memes interactúan durante el proceso de optimización. Ong et al. [286] propusieron una clasificación de los MAs en función de la forma en que los memes se coordinan:

- *Hiperheurística adaptativa*: la coordinación de los memes se realiza a través de reglas heurísticas (ver en [49, 80, 191, 197]). Esto incluye los algoritmos cuyos memes se coordinan por medio de esquemas predeterminados (deterministas o aleatorios). En los aleatorios los memes se van activando uno a uno aleatoriamente en un orden determinado mediante una regla [58]; en cambio, en los deterministas existe una planificación previa [197]. Básicamente se trata de una coordinación de memes sin adaptación. El espacio de decisión se divide en subáreas diferentes cada una con un optimizador distinto y con memes diferentes.

- *Aprendizaje meta-lamarckiano*: los memes tienen definida una probabilidad de activación, permitiendo su adaptación de forma dinámica haciéndolos sumamente flexibles para diversos problemas [220, 265, 284]. En los MAs meta-lamarckianos, el conjunto de memes candidatos compiten entre sí, basándose por ejemplo en las mejoras conseguidas por cada uno de ellos en el pasado y en función de este criterio se decide qué memes progresan y evolucionan y cuáles desaparecen por completo de la población.

- *Adaptación a la diversidad*: se utiliza una medida de la diversidad de la población (por ejemplo la entropía de Shannon) para seleccionar y activar los memes más apropiados [55, 264]. El MA automáticamente coordina los memes analizando el estado de la población completa y más concretamente su diversidad.

- *Auto-adaptación y Coevolución*: en la auto-adaptación los memes se codifican conjuntamente con las soluciones formando parte del proceso evolutivo (se incluyen en los procesos de selección, recombinación, mutación, etc.) [207, 331, 369] y su acción se aplica directamente a la solución huésped; en definitiva, cada solución está compuesta de material genético y memético. Por otra parte, en la coevolución los memes no están directamente conectados con genes sino que coexisten en poblaciones separadas y evolucionan en paralelo con ellas. Para su aplicación y cómputo del fitness se define un mecanismo para enlazarlos con genes concretos, pero dicho mecanismo no es permanente por lo que un mismo meme puede aplicarse a genes diferentes.

Nótese que en sí misma, la idea de gestionar explícitamente los memes está enraizada en los trabajos iniciales del campo de los MAs. Así Moscato [255] concluyó que los memes pueden actuar en dos niveles diferentes –o en dos escalas temporales: a corto plazo, un conjunto de agentes se encontrarían explorando el espacio de búsqueda del problema considerado; en cambio, en el largo plazo, los memes podrían dedicarse a adaptar sus estrategias de búsqueda al escenario actual en el que se encuentre el proceso [60, 61, 285, 286].

Los memes se pueden agrupar formando grupos de memes más complejos o *memeplexes*. Al igual que los genes complejos que se pueden encontrar en biología, los memeplexes son grupos de memes que se pueden encontrar también en

un mismo individuo [60]. En el marco de la MC los memeplexes son un paradigma autoconsistente para resolver problemas en el que memes con características comunes se agrupan para mejorar el aprendizaje y por lo tanto la eficiencia en el proceso de búsqueda. Los memeplexes se pueden ver también como una red memética dinámica, evolutiva y distribuida en la que los memes se complementan unos con otros. Desde un punto de vista estructural, los memeplexes son un conjunto estable y coadaptado en el que los memes que lo contienen se ayudan mutuamente [86]. Según Chen y Ong [60] los memeplexes toman la forma de una estructura de cableado neuronal siendo el cerebro de los agentes el que controla su comportamiento. Más formalmente para cada solución s_τ se asocia un memeplex $\mathcal{M}_\tau = m_1 \dots m_k$ donde cada meme individual m_i codifica la unidad de refinamiento local del proceso de aprendizaje (considérese que no todos los memes de un memeplex tiene que estar activos en el mismo instante de tiempo). Durante el tiempo de vida del proceso de aprendizaje, un memeplex $\mathcal{M}$ mejora una solución s a través de los memes que codifican los caracteres de refinamiento local de s. Durante el tiempo de aprendizaje los memeplexes se activan y desactivan siguiendo un proceso de selección similar al de los EAs y pudiendo sufrir también un proceso de recombinación/mutación propio.

En la línea de todo lo anterior, los MMAs se pueden definir como una especialización de los MAs donde los memes se añaden de forma explícita al genotipo de los individuos formando parte del proceso evolutivo junto con los genes. Fueron inicialmente introducidos por Krasnogor et al. [200, 206] siendo la diferencia fundamental con respecto a los MAs la utilización de una familia de operadores de búsqueda local. Mientras que un MA utiliza un único mecanismo de búsqueda local –aunque pueda ser complejo en su funcionamiento–, un MMA emplea un conjunto de operadores diferentes para realizar este proceso de mejora local. En la literatura del campo se han especificado varios modelos para intentar adaptar los memes de forma dinámica durante el proceso mismo de optimización (auto-adaptación). Esta auto-adaptación puede consistir en simples punteros a operadores de búsqueda local existentes (como en las hiperheurísticas o en el aprendizaje meta-lamarckiano), una parametrización determinada para un esquema de búsqueda concreto [202, 264], un sistema de reglas que alteren la estructura del genoma [331] o incluso mecanismos más complejos como la definición de una gramática para crear un nuevo operador de búsqueda local [201, 204].

También se han estudiado las denominadas metaheurísticas auto-generadas [200, 203], las cuales son capaces de crear sus propios operadores de búsqueda local y coevolucionar con el objetivo de resolver el problema considerado. En los MAs auto-generados coexisten dos procesos evolutivos. Por un lado, la evolución tiene lugar a nivel cromosómico como en cualquier otro EA (los cromosomas y los genes representan soluciones del problema que se está intentando resolver) y por otro la evolución también ocurre a nivel memético donde el comportamiento de los individuos o agentes puede modificar las tasas de supervivencia de sus cromosomas. Se ha demostrado que el concepto de los MAs auto-generados se puede implementar obteniendo resultados claramente satisfactorios en numerosos

problemas [200, 203, 332]. En este contexto, los memes se pueden representar como conjuntos de reglas, programas, heurísticas o cualquier otro concepto que pueda servir para mejorar el propio fitness de los individuos sobre los que se aplican. Las interacciones entre los memes y los genes no son directas, sino que los memes alteran los genotipos de aquellos individuos que los contienen. Existen tres fenómenos que el MA auto-generado intenta incorporar para producir sus operadores de búsqueda local que son propios de la evolución cultural (memética) [133], como son el conocimiento específico del problema, la imitación y la simulación. Además, la representación de los operadores a más bajo nivel incluye características tales como la estrategia de aceptación (e.g., búsqueda ascendente, camino aleatorio, etc.), el número máximo de vecinos que se va a muestrear, el número de iteraciones que va a durar la ejecución del algoritmo o una función de decisión que debe ser capaz de determinar si tiene sentido aplicar la heurística a una solución particular o a una región concreta del espacio de búsqueda.

2.5. Propiedades Self-⋆

Como se ha visto en la sección anterior un diseño adecuado incide directamente en la capacidad de la metaheurística empleada para resolver el problema considerado. Una forma de mitigar el esfuerzo necesario para realizar el diseño correcto es trasladar una parte de dicho esfuerzo hacia la técnica metaheurística misma. Esto se puede hacer de diferentes formas y afectando a diferentes capas de la metaheurística. Quizá la manera más intuitiva de hacerlo sea la que involucra a la parametrización del algoritmo, es decir, a los valores de sus parámetros, como por ejemplo a las tasas de aplicación de los operadores [85]. En las siguientes subsecciones se va a abordar en primer lugar el concepto de la auto-adaptación en general y a continuación se definirán formalmente las distintas propiedades que puede tener un sistema para que pueda considerarse auto-adaptado (propiedades self-⋆). Finalmente se incluirá la auto-adaptación en el marco de la EC y de la MC.

2.5.1. Concepto y Necesidad

El concepto de computación autónoma se refiere a la capacidad de los sistemas de computación para auto-gestionarse[43] [182] y busca mejorar estos sistemas limitando la intervención humana en los mismos. Conforme estos sistemas se han ido haciendo cada vez más complejos y sofisticados también ha crecido en la misma proporción la dificultad para gestionarlos e incluso para parametrizarlos adecuadamente con el objetivo de resolver un problema particular, lo que ha contribuido en los últimos años a que se incremente el interés para que los sistemas sean capaces de realizar cada vez más operaciones por sí mismos sin necesidad

[43] *self-managing*

de recurrir a operadores humanos. Así surge el concepto de computación autónoma [194] (y posteriormente el de propiedades self-⋆). Además, el interés por la eficiencia a diversos niveles (diseño e implementación, gestión y explotación) ha incrementado todavía más el interés por los sistemas autónomos debido a que en muchos casos la única vía para conseguir sistemas eficientes es a través de las propiedades self-⋆ [27]. Realmente las propiedades self-⋆, aunque identificadas mediante otros términos, han sido un campo de interés para los científicos y los ingenieros desde que se diseñó y construyó el primer computador.

La situación actual es que los sistemas de computación se han extendido sobre disciplinas muy diversas dando como resultado una amalgama de sistemas, en algunos casos incluso con nexos de unión muy difusos, lo que hace complicado en estos casos encontrar el camino para diseñar y aplicar estas propiedades self-⋆ a dichos sistemas. En cualquier caso es un reto complejo pero a la vez fascinante y supone un aliciente adicional para muchos investigadores que han encontrado en este campo el punto hacia el que focalizar sus investigaciones.

El término "autónomo" procede del ámbito de la biología, donde tenemos muestras claras de sistemas que no requieren la atención del individuo para que se ajusten dinámicamente a cada escenario. Horn en su artículo sobre computación autónoma[44] [180] compara los sistemas de computación con el cuerpo humano, el cual tiene mecanismos (el sistema nervioso central) para controlar de forma autónoma múltiples funciones sin necesidad de que el individuo tenga que ocuparse de ellas conscientemente. Ejemplos concretos los tenemos cuando el cuerpo ajusta su respiración o los latidos del corazón en función del esfuerzo realizado o cuando el proceso digestivo tiene lugar, sin necesidad de requerir un control explícito de nuestro sistema nervioso. Siguiendo este mismo modelo, la computación autónoma, cuyo origen procede de IBM en el año 2001 para describir sistemas de computación auto-gestionados [180], intenta intervenir en los sistemas de computación de forma similar a como sucede en la biología. El objetivo final deseado es que los sistemas de computación complejos tengan propiedades autónomas (self-⋆) para realizar tareas de mantenimiento o de optimización sin necesidad de recurrir a los administradores de sistemas. En este sentido se han perfilado cuatro propiedades fundamentales que deberían tener los sistemas auto-gestionados: auto-configuración[45], auto-optimización[46], auto-curación[47] y auto-protección[48]. A este tipo de atributos se les conoce como propiedades self-⋆. Sobre estas y otras propiedades trata el resto de la sección.

[44] *autonomic computing*
[45] *self-configuration*
[46] *self-optimization*
[47] *self-healing*
[48] *self-protecting*

2.5.2. Formalización

A pesar del auge creciente por las propiedades self-⋆ existe una gran ambigüedad en el uso de estos términos. Berns y Ghosh propusieron una nomenclatura que permite resolver esta ambigüedad formalizando todos estos conceptos a través de definiciones precisas [41]. En esta sección se pretende utilizar dicho formalismo para explicar todas estas propiedades. En primer lugar hay que establecer el marco conceptual sobre el que se van a definir las propiedades self-⋆.

Definición 23. *Se consideran n procesos ($n \geqslant 1$) y un grafo no dirigido $G = (V, E)$ que representa la topología existente entre los n procesos, siendo V el conjunto de procesos y E las interconexiones entre ellos. Cada proceso $\pi_i \in V$ tiene un conjunto $\mathcal{N}(\pi_i) \subseteq V$ de procesos vecinos, es decir, $(\pi_i, \pi_j) \in E \Leftrightarrow \pi_j \in \mathcal{N}(\pi_i) \wedge \pi_i \in \mathcal{N}(\pi_j)$.*

Definición 24. *Cada proceso π_i ejecuta un programa que consiste en un número finito de acciones del tipo $\gamma \rightarrow \eta$, donde γ es un predicado sobre todas las variables de π_i y $\mathcal{N}(\pi_i)$ y η es una acción que actualiza el estado local $\theta(\pi_i)$ de π_i.*

Definición 25. *El estado global Θ se denomina* configuración *y consiste en el conjunto de los estados locales de todos los procesos, es decir, $\Theta = \{\theta(\pi_i) \mid \pi_i \in V\}$.*

A partir de las definiciones anteriores se puede definir el concepto de *computación* como sigue:

Definición 26. *Una computación es una secuencia finita o infinita de estados globales que satisfacen que si Θ y Θ' son dos estados consecutivos en una secuencia dada entonces existe un proceso π_i que tiene una acción en Θ y la ejecución de dicha acción lleva al proceso al estado Θ'.*

El conjunto de todas las posibles acciones se divide en dos subconjuntos: interno y externo. Una acción interna es una acción del sistema descrita como en una de las definiciones previas (Definición 24). En cambio una acción externa tiene un sentido más amplio: por un lado es una acción causada por un proceso diferente y por otro puede alterar el *entorno* R. Se entiende por entorno el conjunto de parámetros que las acciones internas pueden leer pero no alterar. Las acciones externas pueden modificar el estado interno de cualquier proceso e incluso provocar fallos, cambiar la configuración global del sistema o incluso cambiar la topología de la red. Sea Υ el conjunto de todas las acciones externas posibles. La correctitud de un sistema viene caracterizada por sus propiedades de seguridad[49] (aquellas que impiden que el sistema se deteriore) y vivacidad[50] (aquellas que eventualmente permiten la ejecución de alguna acción, es decir, que permiten que el sistema progrese) [16]. Las propiedades de seguridad se caracterizan por el invariante Φ.

[49] *safety*
[50] *liveness*

Definición 27. *Un sistema se encuentra en una configuración consistente respecto a un entorno dado R cuando su predicado de seguridad correspondiente Φ se cumple.*

Se han definido cuatro tipos de tolerancias en función de cómo las acciones externas influyen en las propiedades de seguridad o vivacidad y cómo estas se reestablecen [23]:

- enmascarada[51]: un sistema tiene tolerancia enmascarada con respecto a una acción C cuando esta acción no afecta a las propiedades del sistema.
- no enmascarada[52]: a diferencia de la anterior, existe alguna acción externa $C \in \Upsilon$ que viola la propiedad de seguridad (pero no la de vivacidad), pudiendo ser restablecida en algunos casos.
- a salvo de fallos[53]: un sistema está a salvo de fallos con respecto a una acción $C \in \Upsilon$ si compromete la propiedad de vivacidad pero no la de seguridad.
- degradación elegante[54]: este caso se da cuando una acción externa afecta a las propiedades de seguridad, pero no a las de vivacidad, y el sistema es capaz de recuperarse a una configuración que satisface una condición más débil de seguridad Φ' ($\Phi \subset \Phi'$), pero que es aceptable para la aplicación.

A continuación se van a definir las diferentes propiedades self-$\star$ basadas en el modelo descrito en los párrafos anteriores.

2.5.2.1. Auto-gestión

Se entiende por un sistema con auto-gestión cuando el sistema tiene al menos una propiedad self-$\star$ [338], es decir, esta es una propiedad general que puede considerarse la raíz de todas las demás. Una definición precisa siguiendo el modelo descrito es:

Definición 28. *Un sistema es auto-gestionado con respecto a un subconjunto de acciones externas $C \subseteq \Upsilon$ si de forma automática, es decir, sin intervención humana, mantiene, mejora o restablece la propiedad de seguridad Φ siguiendo la secuencia de acciones C.*

La definición anterior es muy genérica debido a que la propiedad de auto-gestión también lo es. Refinando esta definición se obtienen las definiciones del resto de propiedades que se van a exponer a continuación.

[51] *masking*
[52] *non-masking*
[53] *fail-safe*
[54] *graceful degradation*

2.5.2.2. Auto-estabilización

La auto-estabilización[55] fue introducida por Dijkstra [93] y ha sido uno de los puntos de investigación más activos durante muchos años. Es un tipo de tolerancia no enmascarada que se refiere a la capacidad de un sistema para recuperarse espontáneamente a partir de cualquier configuración inicial.

Definición 29. *Para que un sistema sea auto-estabilizado debe cumplir las siguientes condiciones:*

1. *si empezando a partir de una configuración inicial arbitraria se recupera a una configuración consistente donde el predicado de seguridad* Φ *se cumple;*

2. *si permanece en esa configuración a partir de ese momento.*

Un ejemplo de problema que cumpla con esta propiedad puede ser un problema con restricciones, en el que partiendo de una población inicial de soluciones que pueden ser no factibles el algoritmo alcanza una población de soluciones factibles tras un número determinado de configuraciones, sin generar soluciones no factibles a partir de ese momento.

2.5.2.3. Auto-curación

Esta propiedad se centra en el mantenimiento y restauración de las propiedades de seguridad de un sistema siendo capaz de detectar problemas y corregirlos [44, 144], por ejemplo, cambiando automáticamente a un componente redundante. Es importante que durante el proceso de corrección del problema no se introduzcan nuevos errores que no existían antes. La tolerancia a fallos (se tratará en un capítulo posterior) es un aspecto importante de la auto-curación.

Definición 30. *Un sistema se dice que dispone de auto-curación con respecto a un conjunto de acciones externas* $C \subseteq \Upsilon$ *si las ocurrencias de las acciones de* C *provocan como máximo una violación temporal de la propiedad de seguridad del sistema* Φ*.*

Un sistema con esta propiedad puede tener una pérdida de rendimiento significativa cuando se produce un fallo múltiple, ya que el tiempo necesario para que el sistema se recupere puede llegar a ser en algunos casos elevado, por ello, hay que elegir cuidadosamente qué aspectos del sistema se desea que cumplan esta propiedad. El uso de operadores de reparación de soluciones infactibles puede ser un claro ejemplo de esta propiedad [70, 247, 248].

[55] *self-stabilization*

2.5.2.4. Auto-organización

El término auto-organización se ha empleado repetidamente en diversas disciplinas científicas. En Ciencias de la Computación hace referencia a una propiedad self-$\star$ con tolerancia de tipo no enmascarada que consiste en la capacidad de un sistema para mejorarse automáticamente de forma que cada proceso tome decisiones en función de sí mismo y de sus vecinos más próximos [19].

Definición 31. *Un sistema con n procesos es auto-organizado*[56] *si mantiene, mejora o restaura una o más propiedades de seguridad* Φ *siguiendo la secuencia de un conjunto de acciones externas* $C \subset \Upsilon$.

Un ejemplo de algoritmo con auto-organización puede ser aquel que mantiene en un rango determinado el nivel de diversidad de la población (por ejemplo usando la entropía de Shannon sobre una medida de distancia definida entre dos genotipos) empleando para ello operadores en función de dicha diversidad [260].

2.5.2.5. Auto-protección

La auto-protección normalmente está dirigida a mitigar los efectos que puedan tener las acciones externas maliciosas sobre el sistema mediante políticas de privacidad o mecanismos de confianza adecuados que aseguren el sistema y sus datos [18, 109]. Normalmente se trata de una tolerancia de tipo enmascarada.

Definición 32. *Un sistema disponde de auto-protección, si de forma continua mantiene una propiedad de seguridad* Φ *en presencia de un subconjunto de acciones externas con intenciones maliciosas* $C \subset \Upsilon$.

Esta definición es bastante genérica y se solapa incluso con la de auto-gestión. No obstante los predicados involucrados en la auto-protección son aquellos relacionados con la integridad del sistema y de los datos. Un buen ejemplo de auto-protección puede ser la utilización de métodos para evitar que un superindividuo (solución con un fitness mucho mejor que la media) se propague por un modelo de islas acabando con la diversidad de las poblaciones [250].

2.5.2.6. Auto-optimización

Un sistema con esta propiedad tiene la capacidad para decidir de forma autónoma si es necesario optimizar el uso de los recursos disponibles de forma proactiva para mejorar el rendimiento global. A diferencia del resto de propiedades self-$\star$, la auto-optimización está relacionada con la función objetivo como propiedad del sistema (en vez de con predicados de seguridad del tipo restauración o mantenimiento) ya que intenta minimizar (maximizar) esta función. El objetivo por lo tanto es mantener el sistema en la configuración óptima.

[56] *self-organization*

Definición 33. *Un sistema es auto-optimizado si comenzando desde una configuración inicial arbitraria mejora (minimiza o maximiza) el valor de una función objetivo predefinida de su estado global.*

Una gran parte de la investigación en EAs auto-adaptativos [333] va en esta línea como se verá en el capítulo siguiente.

2.5.2.7. Auto-configuración

La auto-configuración es una propiedad que permite a un sistema cambiar sus parámetros de funcionamiento en función de ciertos objetivos de más alto nivel sin necesidad de especificarle cómo debe hacerlo.

Definición 34. *Un sistema es auto-configurado con respecto a un conjunto de acciones $C \subset \Upsilon$ si es capaz de cambiar su configuración para restaurar o mejorar alguna propiedad de seguridad Φ definida sobre el espacio de configuraciones.*

En cierto modo la auto-optimización puede verse como una auto-configuración dinámica. Normalmente se trata de tolerancia de tipo no enmascarada.

2.5.2.8. Auto-escalado

El auto-escalado[57] es una propiedad involucrada en la capacidad de un sistema para responder eficientemente ante tamaños del problema diferentes y también hace referencia a aquellos sistemas que son capaces de cambiar su tamaño de forma dinámica [236, 373].

Definición 35. *Un sistema es auto-escalado[58] con respecto a un conjunto de acciones externas que afectan a sus parámetros de escala $C \subset \Upsilon$ si mantiene o mejora una propiedad del sistema Φ durante la ocurrencia de estas acciones.*

En este caso, un ejemplo claro puede ser un algoritmo que sea capaz dinámicamente de cambiar el tamaño de la población en función de algún criterio [120, 121] (se pueden introducir nuevos individuos cuando la diversidad ha llegado a un cierto nivel o cuando las tasas de mejora de los operadores –fitness de la descendencia con respecto al fitness de los padres– se han estancado).

2.5.2.9. Auto-inmunidad

La auto-inmunidad[59] es una propiedad relativamente nueva y poco estudiada [41] cuyo objetivo es dotar a los sistemas de cierto nivel de aprendizaje que les permita tomar decisiones adecuadas en función de cada escenario posible en vez de tomar estas decisiones de forma estadística. El hecho de disponer de un sistema con aprendizaje permite mitigar la sobrecarga que generan en ocasiones los

[57] *self-scaling*
[58] *self-scaling*
[59] *self-immunity*

mecanismos de tolerancia a fallos guiados exclusivamente por técnicas estadísticas.

Definición 36. *Un sistema es auto-inmune con respecto a un conjunto de acciones externas $C \subseteq \Upsilon$ si*

1. *restaura los predicados de seguridad a partir de las acciones de C.*
2. *los predicados de seguridad no son puestos en peligro a pesar de que existan acciones maliciosas en C.*

Un ejemplo de esta propiedad se encuentra en [352], donde se estudia cómo un EA distribuido es capaz de recuperarse ante actos maliciosos a través de mecanismos de tolerancia a fallos.

2.5.2.10. Auto-contención

La auto-contención[60] es una característica que poseen aquellos sistemas que son capaces de limitar el impacto de acciones maliciosas a un número reducido de los procesos existentes.

Definición 37. *Un sistema es auto-contenido si acciones externas maliciosas $C \subseteq \Upsilon$ ponen en peligro tan solo una fracción de los procesos de un sistema distribuido.*

Un ejemplo de auto-contención lo podemos encontrar en una red cuando uno de los nodos ha sido atacado por algún proceso externo malicioso; en este caso, si los demás nodos son capaces de detectarlo podrían desconectarlo de la red para que el resto de nodos no se vean afectados.

2.5.3. Propiedades Self-$\star$ en Computación Evolutiva y Memética

Uno de los principales retos de la EC y de la MC es decidir la parametrización adecuada para cada algoritmo y problema, ya que esta cuestión tiene un impacto considerable en el rendimiento posterior de las metaheurísticas en general y de los EAs y MAs en particular. Ya en los inicios de la EC, Schwefel propuso usar los principios de la auto-adaptación[61] [326] para ajustar los parámetros de control de los EAs de forma automática y durante el proceso mismo de optimización. Esto supone una diferencia importante con respecto al mecanismo más tradicional de ajustar los parámetros de forma predeterminada o de cambiarlos siguiendo criterios fijos y planificados de antemano, ya que en este caso dichos parámetros de control del algoritmo forman parte del propio proceso evolutivo (incluso podrían

[60] *self-containment*
[61] *self-adaptation*

formar parte de la codificación de las soluciones), es decir, pueden evolucionar junto con el resto de características de los individuos [42].

La configuración de los parámetros se puede realizar de dos formas diferentes, bien de forma estática, es decir, ajustando estos parámetros sobre ejecuciones previas del algoritmo y posteriormente lanzando el algoritmo con los parámetros definitivos –para ello se podrían utilizar los denominados Algoritmos Genéticos Virtuales[62], los cuales consisten en simulaciones de GAs reales en las que se ha eliminado la carga del cálculo de la función de fitness y de aplicación de los operadores sustituyéndolos por operaciones virtuales basadas en un modelo estadístico previamente construido [164, 266–268]–, o bien de forma dinámica, esto es, el algoritmo comienza su ejecución con un conjunto determinado de parámetros y durante su ejecución los va modificando de alguna manera. La forma estática se ha denominado habitualmente en la literatura científica como "ajuste de parámetros"[63], mientras que la segunda se suele denominar "control de parámetros"[64]. El ajuste de parámetros se realiza de forma experimental con diferentes valores para cada parámetro, seleccionando aquellos que conducen al algoritmo a mejores resultados. Evidentemente, cuanto mayor sea el conjunto de parámetros más complejo es este tipo de búsqueda de los parámetros adecuados y además el consumo de tiempo para realizarlo puede crecer de forma exponencial. Algunas de las desventajas de hacerlo de esta forma se enumeran a continuación:

- Los parámetros no son independientes, por lo que realizar este proceso de forma sistemática puede ser algo prácticamente imposible.
- Los EAs son técnicas estocásticas y por lo tanto, para establecer diferencias estadísticamente significativas hay que ejecutar múltiples pruebas de cada combinación de parámetros.
- El consumo de tiempo puede ser muy elevado, sobre todo conforme crece la cardinalidad del conjunto de parámetros.
- No es posible asegurar que para un problema considerado los valores de los parámetros elegidos sean realmente los óptimos.

Por otro lado, el ajuste dinámico de los parámetros (control de parámetros) de un EA o de un MA en general puede considerarse una propiedad del tipo auto-configuración –aunque realmente todos los EAs y los MAs que implementan propiedades self-⋆ tienen como objetivo último la auto-optimización– y puede realizarse según diversas estrategias, las cuales se pueden clasificar de acuerdo a Angeline y Hinterding [21, 173] de la siguiente manera:

- *Control de parámetros determinista*: existe una estrategia planificada y predefinida con los cambios en los parámetros que se van a realizar durante la ejecución del EA.

[62] *Virtual Genetic Algorithms*
[63] *parameter tuning*
[64] *parameter control*

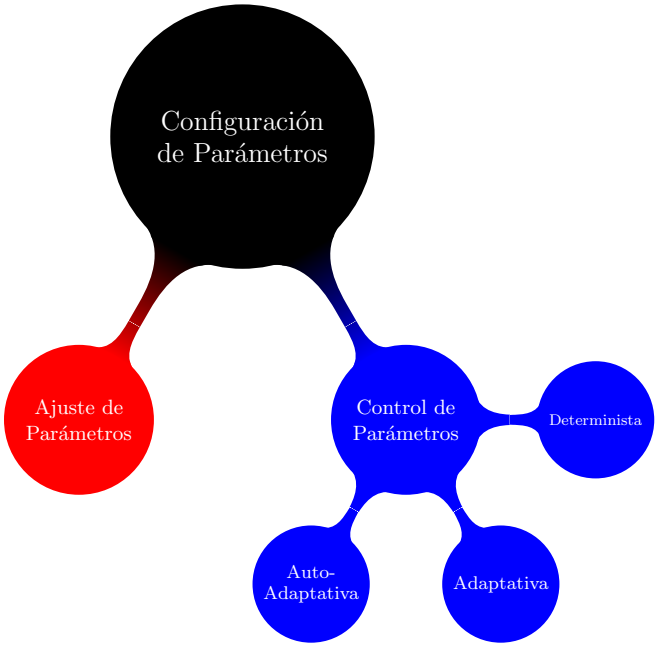


Figura 2.6: Resumen de técnicas para configurar los parámetros en un EA y en un MA.

- *Control de parámetros adaptativo*: se dispone de un mecanismo retroalimentado que en tiempo de ejecución permite ajustar los parámetros de control en función de si se producen mejoras o no en la función objetivo.
- *Control de parámetros auto-adaptativo*: los valores de los parámetros de control evolucionan junto con las características de los individuos aplicando operadores evolutivos específicos para ello, es decir, se aprovecha el proceso competitivo propio de los EAs para determinar si los cambios en los parámetros provocan efectos beneficiosos o no en el fitness de los individuos de la población.

En la Figura 2.6 se puede ver un resumen con las principales técnicas para configurar los parámetros de un EA o un MA. A partir de la clasificación anterior, se puede distinguir entre dos tipos diferentes de auto-adaptación en función del tipo de evidencias encontradas para realizar cambios en los parámetros:

- *Evidencias absolutas*: cuando una regla modifica el valor de un parámetro en función de la ocurrencia de un evento predefinido (este evento se dispara en función de criterios basados en el estado de la búsqueda, a diferencia del control de parámetros determinista donde el evento lo desencadena un evento determinista, como por ejemplo el tiempo o las generaciones transcurridas).
- *Evidencias relativas*: la modificación en los parámetros depende de otras variables, como por ejemplo el fitness de la descendencia.

Algunas de las características que debe exhibir un EA para considerarse auto-adaptativo se pueden resumir en las siguientes:

- El algoritmo debe disponer de un control automático de sus operadores y de los parámetros involucrados en ellos, de forma que el proceso evolutivo los haga evolucionar junto con las soluciones. Esto implica que cada individuo debe codificar sus propios parámetros.
- Debe haber un conjunto de parámetros codificados en cada individuo para que la evolución disponga de la suficiente diversidad y la auto-adaptación sea verdaderamente efectiva.
- Para que el proceso evolutivo sea satisfactorio debe poder establecerse una relación entre los parámetros codificados y los cambios producidos en la codificación del problema, al igual que entre las acciones de un operador y los posteriores cambios en el fitness de la solución codificada o entre el fitness de un individuo y la selección previamente ejecutada.

La auto-adaptación como se ha visto anteriormente puede centrarse en múltiples parámetros del algoritmo, tanto en parámetros generales de control global como en parámetros particulares involucrados en aspectos concretos del mismo. Esto último puede resumirse principalmente en dos aspectos de un MA:

- *Mutación y recombinación*: se puede realizar un ajuste dinámico o auto-adaptativo de los parámetros involucrados en la mutación y en la recombinación. Algunos parámetros que se pueden auto-ajustar son las probabilidades de mutación y recombinación o el tipo de operador aplicado en cada caso. Si el algoritmo tiene la capacidad de modificar en tiempo de ejecución estos parámetros se incrementa la probabilidad de que haya un cambio en la región de exploración dentro del espacio de búsqueda lo que en muchas ocasiones puede beneficiar al propio algoritmo en términos de eficiencia a través de un incremento de la diversidad. Esto es particularmente importante en el caso de las ESs [324, 325]. En ellas el operador de mutación está basado en una distribución Gaussiana, la cual requiere dos parámetros: la media (μ) y la desviación estándar (σ), aunque en la práctica $\mu = 0$ por lo que solo es necesaria la desviación estándar. La mutación se aplica a cada componente del vector $\vec{x}$ de forma independiente produciendo $x'_i = x_i + N(0, \sigma)$, donde $N(0, \sigma)$ es un número aleatorio obtenido a través de la distribución Gaussiana con media cero y desviación estándar σ. El valor de σ se denomina habitualmente como "tamaño del paso". La base de la auto-adaptación en las ESs es que el tamaño del paso está incluido en los cromosomas y por lo tanto está implícitamente sometido a los procesos de selección y variación, lo que tiene como consecuencia que el tamaño del paso varíe durante la ejecución del algoritmo y es el proceso evolutivo en sí mismo el que ajustará este valor adecuadamente a cada momento y cada individuo. Estos métodos de mutación y recombinación se pueden considerar como auto-optimización.

- *Búsqueda local*: en el caso de los MAs existe al menos una operación adicional de búsqueda local y por lo tanto, los parámetros relacionados con ella son también apropiados para incorporarlos al proceso general de auto-adaptación (en este caso también podría hablarse de auto-optimización), incluso pudiendo cambiar el método de búsqueda local en función de determinados criterios, es decir, se puede disponer de un conjunto de operadores de búsqueda local de forma que el MA puede elegir cuál utilizar en cada momento o por cada individuo. Una elección adecuada del operador de búsqueda local es decisiva en la eficiencia del MA. Por ello, numerosos autores han investigado cómo introducir mecanismos para poder elegir de forma dinámica entre un número determinado de operadores de búsqueda local. Algunos ejemplos de ello los tenemos en los MMAs [200, 205], el modelo COMA de Smith [330, 331], los MAs meta-lamarckianos de Ong y Keane [284] o las hiperheurísticas [48, 49, 80, 191]. Otro modelo interesante es el de la auto-generación de estrategias de búsqueda local de Krasnogor y Gustafson que utiliza una gramática para determinar por ejemplo cuándo tiene lugar la búsqueda local [201, 204].

Como se mencionó anteriormente, una gran parte de la literatura de este campo está enfocada hacia la auto-adaptación en los operadores de variación, e.g. [116, 117]. Sin embargo, existen otros factores igualmente importantes para el rendimiento de un EA, como el proceso de selección, que no se han estudiado con tanta frecuencia. Sí cabe mencionar el trabajo de Fernandes y Rosa [118] en el que en vez de emplear un proceso de selección aleatorio lo hacen en función de la semejanza entre los progenitores, definiendo para ello una medida de distancia entre genotipos o fenotipos que se utiliza para decidir los individuos que formarán parte del proceso de recombinación. Por su parte, Eiben et al. [105] analizaron la influencia del tamaño de la población sobre el rendimiento del EA pero ampliándolo al proceso de selección también. Un EA capaz de adaptar el tamaño de su población cumple la propiedad de auto-escalado. En esta línea se han introducido mecanismos de auto-adaptación para intentar mitigar los efectos de una elección inicial inadecuada del tamaño de la población. Se ha estudiado el efecto que tienen los cambios del tamaño de población en GP utilizando el concepto de plaga (una analogía de la presencia de individuos enfermos o infectados, que a efectos computacionales se implementa eliminando un determinado número de individuos de la población de forma periódica –cada cierto número de generaciones) como mecanismo de auto-adaptación del algoritmo [120, 121]. Aunque las plagas tienen connotaciones negativas en biología en el ámbito de la computación pueden ser un elemento que incremente la eficiencia en el uso de los recursos computacionales a la vez que afecta a la tasa de supervivencia de los individuos de la población. Estos algoritmos cumplen la propiedad de auto-configuración.

Por otra parte, la auto-adaptación se puede ampliar a otros modelos de EAs como los EAs celulares, los distribuidos o aquellos que definen una topología dinámica. En cuanto a los EAs celulares se ha estudiado la auto-adaptación de la

topología de conexión a través de la definición de regiones de vecindad de forma tal que los individuos solo pueden interactuar con sus vecinos [7]. Alterando tanto la forma de la topología como el vecindario se pueden generar múltiples variantes en el proceso de búsqueda. Para ello se define una relación (ratio) entre la forma de la topología y del vecindario y se analiza la influencia de este ratio en la compensación entre explotación y exploración durante la ejecución del algoritmo.

Con respecto a las estructuras de población dinámicas, estas han tenido un interés notable para la comunidad científica en los últimos tiempos con propuestas que caracterizaban la auto-organización a través de la definición del concepto de localidad y restringiendo las posibles interacciones entre individuos en función de dicha localidad con el objetivo último de mantener la diversidad [361]. Un modelo especialmente interesante es el definido por Fernandes et al., que combina ideas de la PSO y de los autómatas celulares con memoria a corto plazo [119]. Para ello utiliza un tipo de comunicación indirecta entre los individuos de forma que las acciones aisladas de uno de ellos pueden influir en otros o incluso en el conjunto de la población[65]. Esta comunicación está basada en unas sencillas reglas de movimiento en una cuadrícula 2D que acaban formando clusters auto-organizados de partículas [114]. Una característica de estos clusters es que son capaces de evolucionar por sí mismos y cambiar su forma, lo que indica que existe algún tipo de orden dinámico. Este modelo fue posteriormente extendido a otras metaheurísticas como los EAs [113, 115] y los MMAs [280].

Las topologías dinámicas también se han estudiado desde el punto de vista de los sistemas descentralizados [253] donde un conjunto de agentes reconfigurables con respecto al conjunto de recursos disponibles son capaces de adaptarse sobre la marcha o incluso agruparse para formar equipos con agentes complementarios entre sí con el objetivo último de mejorar el rendimiento final del algoritmo. El uso de topologías dinámicas cumple la propiedad de auto-organización.

En EAs distribuidos también es de un interés considerable el estudio sobre la escalabilidad (y cómo afectan al rendimiento del algoritmo estos cambios) y la topología de la red de interconexión, por lo que se han analizado los efectos que causan las comunicaciones entre los nodos cuando el algoritmo se escala. En redes P2P se ha estudiado mediante la definición de un EA con estructura espacial consistente en una población de agentes evolutivos [215] o directamente analizando diferentes topologías sobre algoritmos con distinto número de nodos con procesos migratorios definidos entre ellos [213]. En algoritmos distribuidos en los que se introduzca tolerancia a fallos [227, 228, 321] se cumplen las propiedades de auto-curación y auto-protección.

En la siguiente sección se va a profundizar en los conceptos involucrados en la computación paralela, ya que en la actualidad gran parte de la investigación en el ámbito de la EC se ha enfocado hacia los EAs en entornos paralelos.

[65] *stigmergy*

2.6. Computación Paralela

La computación paralela es un modelo de computación que permite reducir el tiempo de ejecución de numerosas aplicaciones aprovechando la disponibilidad de una colección de dispositivos de cómputo. Durante los últimos años ha surgido un número considerable de arquitecturas de computación paralela, las cuales ya no se aplican exclusivamente a grandes computadores como en los primeros años de estas, sino a una miríada de productos, tales como ordenadores personales, portátiles, notebooks o incluso tabletas y teléfonos móviles[66] que disponen de varios procesadores para ejecutar sus sistemas operativos y aplicaciones. También las Unidades de Procesamiento Gráfico (GPUs)[67] han sido uno de los focos de estudio para la aplicación de las arquitecturas paralelas, debido sobre todo a una demanda creciente de aplicaciones gráficas de alto rendimiento. Por último, las soluciones comerciales en el campo de las Tecnologías de la Información a menudo requieren el uso de clusters con cientos o miles de Unidades Centrales de Proceso (CPUs)[68], así como últimamente la computación en la nube[69], de la que se hablará posteriormente y que proporciona un medio económicamente asequible de computación con grandes ganancias de rendimiento.

Las arquitecturas paralelas tradicionales constaban de un conjunto de elementos con características similares, lo que se ha denominado habitualmente como arquitecturas homogéneas. Sin embargo, en los últimos años y debido fundamentalmente a las necesidades tanto científicas como de la industria, estos sistemas homogéneos se han ido ampliando de forma progresiva introduciendo nuevos elementos (procesadores, memorias, dispositivos de red, etc.) diferentes de los anteriores dando lugar a arquitecturas heterógeneas.

Lo anterior es aplicable en lo relativo al hardware, pero a nivel algorítmico también ha habido una evolución. Es bien sabido que el rendimiento es más sensible a un buen diseño del software que al propio hardware. Por ello, se ha hecho necesario el diseño y desarrollo de nuevos algoritmos que sean capaces de ejecutarse eficientemente en arquitecturas paralelas y los algoritmos evolutivos son un buen ejemplo de ello.

En las siguientes secciones se introducirán en primer lugar las arquitecturas paralelas, se discutirán posteriormente algunas ventajas de estas arquitecturas y se finalizará explicando algunos modelos evolutivos paralelos y algunos métodos para medir el rendimiento de los algoritmos paralelos.

2.6.1. Arquitecturas de Computación Paralela

La arquitectura de computadores usada tradicionalmente estaba basada en el conocido modelo ideado por Eckert y Mauchly y descrito por John von Neu-

[66] *smartphones*
[67] *Graphics Processing Units*
[68] *Central Processing Units*
[69] *cloud computing*

mann [353], que consiste en una CPU y memoria donde se ejecuta una única secuencia de instrucciones almacenada en dicha memoria. Esto hace que la limitación de rendimiento venga impuesta por dos factores: por un lado la velocidad a la que se pueden procesar las intrucciones y por otro el rendimiento en cuanto al intercambio de información entre la CPU y la memoria. Desde el primer computador que se construyó siguiendo estas ideas en 1947 (el EDVAC[70]) hasta nuestros días tanto la Ingeniería Informática en general como las Ciencias de la Computación en particular han progresado enormemente. Hoy en día es necesario recurrir a aplicaciones que requieren manejar gran cantidad de información (datos) y realizar numerosas operaciones costosas en tiempo real (instrucciones). Desafortunadamente, a partir de cierta escala en el tamaño de la tarea no existen procesadores capaces de resolver estas cuestiones en un tiempo razonable. Con respecto a esto último, los problemas y la información que utilizan está llegando a ser cada vez más y más compleja. En este sentido se introdujo el término *Big Data* [25, 335] –muy de moda hoy en día– que hace referencia al uso de gran cantidad de datos y que requiere por lo tanto un gran poder de computación para procesarlos adecuadamente. Todo esto implica que es necesario buscar un nuevo modelo de computación. Al margen de diseñar procesadores más rápidos, la alternativa más razonable es idear una arquitectura que permita unir de alguna manera diferentes procesadores para que cooperen entre sí para poder resolver un determinado problema en un tiempo menor al utilizado en caso de usar un único procesador secuencial. En este caso el problema considerado se divide en subproblemas cuya resolución se planifica en los diferentes procesadores disponibles. Evidentemente es necesario establecer mecanismos de comunicación entre los diferentes procesadores para que cada uno de los resultados parciales puedan agruparse y obtener el resultado total correspondiente al problema completo. Este paradigma se denomina "computación paralela" [2, 179, 209, 366].

Los sistemas de computación paralela se han clasificado en función de innumerables criterios aunque quizá uno de los más utilizados es la tradicional taxonomía de Flynn [123] en función del flujo de datos y de las instrucciones, tal y como se muestra a continuación:

- *SISD*[71]. Corresponde al caso secuencial ya que se dispone de un único flujo de instrucciones y un único conjunto de datos.

- *SIMD*[72]. Hay un único flujo de instrucciones pero se opera simultáneamente sobre varios conjuntos de datos.

- *MISD*[73]. Se ejecutan varios flujos de instrucciones al mismo tiempo pero todos gestionan el mismo conjunto de datos.

[70] *Electronic Discrete Variable Automatic Calculator*
[71] *Single Instruction Single Data*
[72] *Single Instruction Multiple Data*
[73] *Multiple Instruction Single Data*

- *MIMD*[74]. Existen múltiples flujos de instrucciones operando a la vez sobre múltiples conjuntos de datos. Es el caso más habitual donde muchos procesadores disponen cada uno de su propia memoria.

En las implementaciones prácticas estos modelos no se siguen de manera exacta, por ejemplo, en los procesadores secuenciales tradicionales (SISD) los datos se agrupan para poder ser accesibles simultáneamente (introduciendo un paralelismo implícito) o mediante segmentación de memoria, dividiendo la ejecución de las instrucciones en diferentes etapas (unidades funcionales), etc. Las arquitecturas MISD tradicionales (multiprocesadores de memoria compartida [346]) fueron las más utilizadas durante la década de 1980 y principios de la de 1990. Sin embargo, fueron las arquitecturas MIMD (multicomputadores con memoria distribuida [304]) las que lideraron la computación paralela durante los años 90, inicialmente mediante estaciones de trabajo[75] conectadas en red. A partir de entonces fueron surgiendo múltiples arquitecturas diferentes de computación paralela, como los circuitos programables (e.g. las FPGAs[76]), los computadores multiprocesador (e.g. los computadores multinúcleo o las GPUs) y las plataformas de computación distribuida que disponen de unidades de cómputo separadas físicamente (e.g. los clusters de computadores o los entornos de computación en grid[77] y computación en la nube). Todo esto ha hecho que en la actualidad la clasificación de Flynn se haya extendido a otros sistemas paralelos como los que se han desarrollado durante estos últimos años. Hoy en día se pueden colocar varios procesadores en un mismo chip lo que ha dado lugar a los *sistemas multinúcleo*, los cuales están disponibles desde hace bastantes años incluso en los ordenadores domésticos (PCs). Estos sistemas son de memoria compartida y se programan utilizando *threads*[78]. En algunos casos especiales, sobre todo en la industria, se utilizan *sistemas empotrados*, los cuales se pueden definir como sistemas de computación que realizan una tarea concreta y normalmente sencilla de forma repetitiva y que proporcionan una respuesta en tiempo real, por lo que habitualmente se diseñan con varios procesadores. Por otro lado la computación en paralelo puede implementarse de muy diversas maneras, a continuación se muestran algunos factores que se pueden tener en cuenta:

- El mecanismo de comunicación entre los diferentes procesadores: normalmente suele ser o bien a través de memoria compartida, es decir, todos los procesadores comparten la misma memoria física o bien mediante memoria distribuida, cuando cada procesador tiene su propia memoria independiente.

[74] *Multiple Instruction Multiple Data*
[75] *workstations*
[76] *Field Programmable Gate Arrays*
[77] *Grid Computing*
[78] hilos o hebras en castellano; son las unidades de procesamiento mínimas que se pueden planificar en un sistema operativo

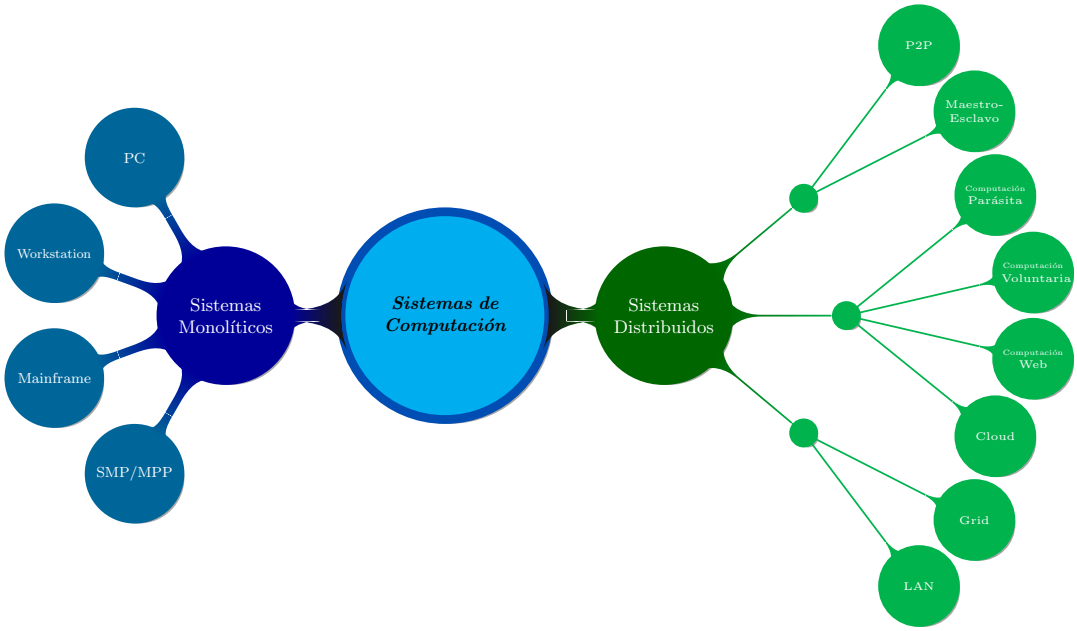
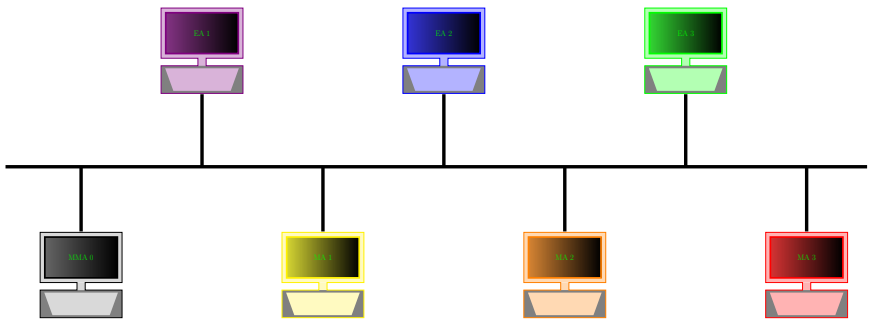


Figura 2.7: Mapa Conceptual de los Sistemas de Computación.

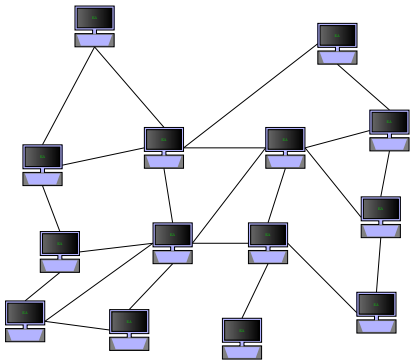
- La red de interconexión: la forma en la que se distribuye la red (topología de red). Algunos ejemplos son en árbol, estrella, malla, etc.
- Si cada procesador ejecuta el mismo algoritmo o no.
- Si la comunicación entre los procesadores se realiza de forma síncrona o asíncrona.

Todos estos elementos hardware combinados con sistemas software diferentes (sistemas operativos, lenguajes y compiladores, etc.) ha hecho que surja un nuevo concepto denominado *virtualización*, que se define como la ejecución simulada en un sistema de computación de un conjunto de tareas diseñadas para realizarlas en otro con características diferentes. La virtualización introduce una capa entre el hardware y el sistema operativo que permite que un mismo usuario pueda utilizar diferentes sistemas operativos a la vez sobre un mismo computador y también permite que múltiples usuarios utilicen máquinas virtuales (simulaciones del comportamiento de un sistema operativo). La virtualización también ayuda a una mejor utilización de los recursos disponibles, ya que normalmente los centros de datos tienen una baja utilización de sus dispositivos de procesamiento y almacenamiento y la virtualización puede mejorar significativamente el aprovechamiento de sus recursos.

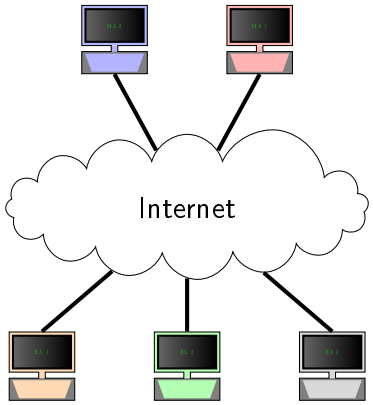
Como se ha visto existe en la actualidad una amalgama de arquitecturas y sistemas con características muy dispares (véase mapa conceptual en la Figura 2.7). De todas ellas son las de computación distribuida las que han ido adquiriendo un peso mayor en el ámbito científico. Estas arquitecturas surgieron originalmente a partir de la incorporación de sucesivos computadores (y dispositivos en general)



(a) LAN



(b) P2P



(c) Cloud

Figura 2.8: Esquemas de diversas arquitecturas de red.

a conjuntos previamente existentes hasta llegar a formar plataformas con cientos o miles de computadores y otros recursos computacionales (almacenamiento, procesamiento gráfico, dispositivos de E/S, etc.) proporcionando la potencia computacional necesaria para resolver problemas complejos en múltiples áreas de conocimiento.

Los sistemas de computación paralela se pueden clasificar en función de múltiples criterios. Si atendemos a la arquitectura física subyacente, los clusters se componen de múltiples computadores independientes conectados a través de una Red de Área Local (LAN)[79] y tienen la capacidad de trabajar de forma cooperativa como si fuesen un único sistema [40, 221]. Además, desde el punto de vista del coste son una alternativa más económica a los sistemas multiprocesador o multicomputador. En la Figura 2.8 (a) se muestra un esquema típico de una red de área local. Debido a su buena relación rendimiento/coste los clusters están

[79] *Local Area Network*

afianzando su posición como sistemas de computación de alto rendimiento[80]. Los clusters pueden estar compuestos por hardware similar en cuyo caso se trata de clusters homogéneos o por elementos diferentes, ya que una de sus grandes ventajas es que se pueden ir ampliando bajo demanda, lo que ocasiona también que las sucesivas incorporaciones de computadores (elementos de red, memoria, procesadores, sistemas de almacenamiento, etc.) sean diferentes a las ya existentes (denominándose entonces clusters heterogéneos [223, 294]). En cualquier caso la ventaja fundamental que los diferencia de los sistemas paralelos analizados hasta ahora es su capacidad para incorporar nuevos elementos conforme va siendo necesario (o eliminarlos si ya han dejado de ser útiles, por ejemplo, si durante un pico de carga se ha necesitado incorporar nuevos computadores y cuando finaliza este procesamiento han dejado de ser necesarios se pueden eliminar del cluster y dedicar a otros procesamientos). Por su parte, la computación en grid hace uso de un conjunto de recursos de computación distribuidos conectados a través de una red y localizados en diferentes lugares geográficos ofreciendo un punto de acceso común a todos ellos. Otra característica típica de los sistemas de tipo grid es que están basados en la interconexión de grupos organizados, unidos entre sí por redes más o menos fiables [40, 221, 372].

Si consideramos el modelo de computación, este puede ser bien centralizado o descentralizado. Entre los primeros es característico el modelo *Maestro-esclavo*[81] que se refiere a una arquitectura en la que un nodo central denominado maestro inicia y controla una sesión con uno o más nodos dependientes del maestro denominados esclavos [15, 32, 39]. Inicialmente se diseñó para redes de computadores *mainframe* donde el mainframe era el computador maestro y los terminales desde donde se conectaban los usuarios eran los esclavos, aunque en general esto último no se cumple siempre hoy en día, ya que los esclavos pueden ser procesadores especializados para ciertas tareas (e.g. cómputos matemáticos). En la Figura 2.9 se muestra un esquema de un modelo típico maestro-esclavo. La *Computación Peer-to-Peer (P2P)*[82] es un ejemplo de modelo descentralizado. Se trata de un paradigma de computación distribuida en el que la carga de trabajo se reparte entre los nodos (pares) disponibles, los cuales se consideran iguales (con los mismos privilegios), es decir, no existe ningún tipo de control central ni de estructura jerárquica [251, 372] (ver Figura 2.8 (b)). Cada nodo a su vez ofrece sus propios recursos al resto de nodos de forma descentralizada. Los nodos que forman parte de un sistema P2P se conectan entre sí mediante redes IP formando una red virtual a nivel de aplicación denominada *overlay* que se utiliza para enrutar los mensajes entre los nodos. Los sistemas P2P se implementan a partir de un gran número de usuarios no fiables con roles similares que se unen entre sí por conexiones también no fiables y que son capaces de ofrecer ciertas características como escalabilidad, resistencia a fallos, almacenamiento o anonimato.

Además de la clasificación anterior, según el origen y uso de los recursos

[80] *High Performance Computing*
[81] *master-slave*
[82] entre pares

de computación existen los denominados sistemas de computación voluntaria, la *Computación en la Web*[83], la *Computación en la Nube* o la *Computación Parásita*[84], por citar solo algunos ejemplos. Los primeros son sistemas en los que los computadores se ponen a disposición de otros usuarios ofreciéndoles sus recursos computacionales para realizar tareas más complejas [320]. Un ejemplo conocido es el proyecto SETI[85], en el que trabajan computadores de todo el mundo para buscar vida extraterrestre. En este caso, SETI sigue un modelo maestro-esclavo aunque también son posibles los modelos P2P. Por otro lado en la computación en la web [127, 132], cuya concepción es mucho más reciente, la propia red se ve como un recurso computacional de manera que se pueden utilizar los navegadores[86] para realizar tareas de computación [242]. De forma similar, la computación en la nube es un modelo de computación paralela en la que recursos de cualquier tipo (computación, software, almacenamiento, etc.) se distribuyen por toda la red y por cualquier localización sin que el usuario sepa dónde están ni qué recurso exacto está utilizando en cada momento (ver Figura 2.8 (c)). Otra característica fundamental de estos sistemas es que los recursos se utilizan bajo demanda, es decir, se adquieren (compran o alquilan) conforme se necesitan y se liberan cuando han dejado de ser necesarios. La virtualización es la base de la tecnología en la nube, ya que usando virtualización los usuarios pueden acceder a servidores o a sistemas de almacenamiento sin conocer los detalles del servidor al que se están conectando o la localización física exacta de sus datos. La capa de virtualización permite al usuario solicitar los recursos computacionales que necesite accediendo a los recursos concretos de forma transparente para el usuario. La computación en la web y la computación en la nube se complementan mutuamente, ya que a menudo permiten reducir la carga computacional del lado del servidor. Por último, la computación parásita es un paradigma de programación a través del cual una aplicación es capaz de completar parte de su carga de trabajo explotando otras aplicaciones [35]. Normalmente el trabajo delegado consiste en costosas operaciones (cálculos complejos) y la interacción entre las aplicaciones se realiza de manera no consentida.

Finalmente todos estos nuevos modelos de computación han dado lugar a lo que se ha denominado como Computación Ubicua, es decir, la posibilidad de que la computación se realice desde cualquier lugar y con cualquier dispositivo.

2.6.2. Computación Distribuida y Algoritmos Bioinspirados

El objetivo fundamental de la computación distribuida es reducir el tiempo de ejecución de un algoritmo mediante la utilización simultánea de múltiples recursos computacionales. En general ejecutar un ciclo completo de un EA para un

[83] *Browser-based Computing*
[84] *Parasitic Computing*
[85] SETI@home (http://setiathome.ssl.berkeley.edu)
[86] *browsers*

problema no trivial, cuando los individuos son complejos o el tamaño de la población es grande requiere una capacidad computacional elevada, principalmente porque la evaluación de la función objetivo suele ser una operación costosa. La gran ventaja de los EAs es que el paralelismo en ellos está incluido en su propio comportamiento, ya que al tratarse con poblaciones cada individuo puede gestionarse como una unidad independiente. Esto proporciona a los EAs una naturaleza intrínsecamente paralela aunque inicialmente no fueran concebidos con esta idea latente. Debido a estas características el rendimiento de los algoritmos basados en poblaciones mejora notablemente cuando se ejecutan en paralelo [4, 12, 237]. Por ello muchos investigadores han abordado el paralelismo implícito de los EAs ya desde la década de 1980, como Grefenstette mediante implementaciones de GAs paralelos [163] o Grosso que introduce un modelo con múltiples poblaciones [165]. Véase [12, 344] para revisiones generales del área.

Con respecto a la medida del paralelismo [158, 190], se ha hablado en ocasiones del *grado de paralelismo* de un algoritmo, esto es, el número de operaciones que pueden realizarse en paralelo (esto no depende tanto del número de procesadores disponibles y sí del algoritmo específico) o de la *granularidad* que es el tamaño de las tareas realizables en paralelo de forma independiente. Los efectos de la paralelización en los EAs se han estudiado ampliamente en diversos trabajos, e.g. en [344], en el que se sugiere que existen diversas formas para calcular el tiempo de computación de un EA paralelo. La más sencilla es medir el tiempo real que transcurre entre el inicio de la tarea y el final[87], aunque bajo ciertas condiciones se podrían utilizar parámetros característicos de los EAs como el número de evaluaciones de la función objetivo o el número de generaciones del algoritmo (considerando de alguna manera el coste adicional producido por los procesos de coordinación entre computadores o los tiempos de comunicación entre ellos).

Según lo visto hasta ahora se puede deducir fácilmente una de las grandes ventajas de los algoritmos evolutivos como es que son fáciles de paralelizar. El proceso de evolución subyacente puede ser implementado fácilmente sobre diversas máquinas a la vez y de diversas maneras diferentes. Se puede tanto implementar la paralelización sobre operaciones concretas como sobre el proceso evolutivo en sí mismo. Para este último caso, se han desarrollado diversas variantes de algoritmos de búsqueda, como los modelos de islas o los algoritmos evolutivos celulares. La principal diferencia con respecto a una implementación clásica es que la evolución tiene lugar sobre una red con estructura espacial. En cada procesador o nodo evoluciona una subpoblación y las soluciones son intercambiadas entre dichos procesadores/nodos en algún momento del proceso a través de un flujo migratorio de individuos entre poblaciones. La cantidad de información –soluciones– intercambiada se puede ajustar mediante algún parámetro, evidentemente cuanto mayor sea esta menor será la diversidad total del conjunto de poblaciones. En general, se ha demostrado que los algoritmos evolutivos paralelos mejoran la eficiencia y la eficacia del proceso de búsqueda, es decir, se consigue encontrar mejores solu-

[87] *wall-clock time*

ciones con un coste computacional menor [3]. A continuación se van a exponer algunas formas de implementar el paralelismo en los EAs.

2.6.2.1. Modelos Maestro-esclavo

La forma más sencilla de implementar EAs paralelos es ejecutar ciertas operaciones en procesadores independientes y tanto los operadores de variación –mutación y cruce– como las funciones de evaluación (de hecho son estas últimas las más apropiadas para ejecutarse en paralelo ya que suelen tener un coste computacional más elevado) son susceptibles de ejecutarse de forma independiente. Este tipo de implementación –arquitectura– se denomina modelo maestro-esclavo[88] y consiste básicamente en que uno de los computadores, el denominado como maestro, distribuya la carga en ejecución sobre el resto de computadores denominados esclavos. Esta arquitectura es muy apropiada para crear las poblaciones de hijos y evaluarlas de forma independiente tras haber seleccionado previamente a los padres correspondientes. Más concretamente, el funcionamiento habitual de esta aproximación consiste en que un procesador central (maestro) realiza las operaciones de selección mientras los procesadores esclavos ejecutan los operadores de variación y la evaluación de la función objetivo. En esencia este modelo tiene un comportamiento similar al de un EA secuencial, pero con una eficiencia computacional más elevada, especialmente por el tiempo consumido por la evaluación de las funciones objetivo que se reparte entre todos los computadores esclavos. Un buen ejemplo de MA que sigue el modelo maestro-esclavo se puede encontrar en [240] donde se resuelve el problema de la ordenación de microarrays de Genes[89] (**NP**-duro), que consiste en la ordenación de conjuntos de genes agrupándolos por comportamientos similares, el cuál se determina evaluándolo en función del nivel de actividad de los genes. Este modelo consigue una ganancia muy grande cuando se implementa en paralelo con respecto a su versión equivalente secuencial debido a que la búsqueda local (ejecutada en los esclavos) consume un porcentaje muy elevado ($> 95\,\%$ en algunos casos) del coste computacional total del algoritmo por lo que es muy ventajoso ejecutarla en paralelo sobre la población (ver Figura 2.9).

Los modelos maestro-esclavo son normalmente síncronos, es decir, el maestro espera a que los esclavos hayan finalizado sus operaciones para proceder a la sincronización, aunque también existen implementaciones asíncronas. Por otro lado, el comportamiento de los modelos maestro-esclavo síncronos no es muy diferente del de sus correspondientes secuenciales: la implementación es diferente, pero el comportamiento de la búsqueda es el mismo. Nótese también que se puede distinguir entre un modelo con evaluación paralela de la población y otro con evaluación distribuida de cada solución:

- *Evaluación paralela de la población*: este modelo es habitualmente más uti-

[88] *master-slave model*
[89] *Microarray Gene Ordering Problem*

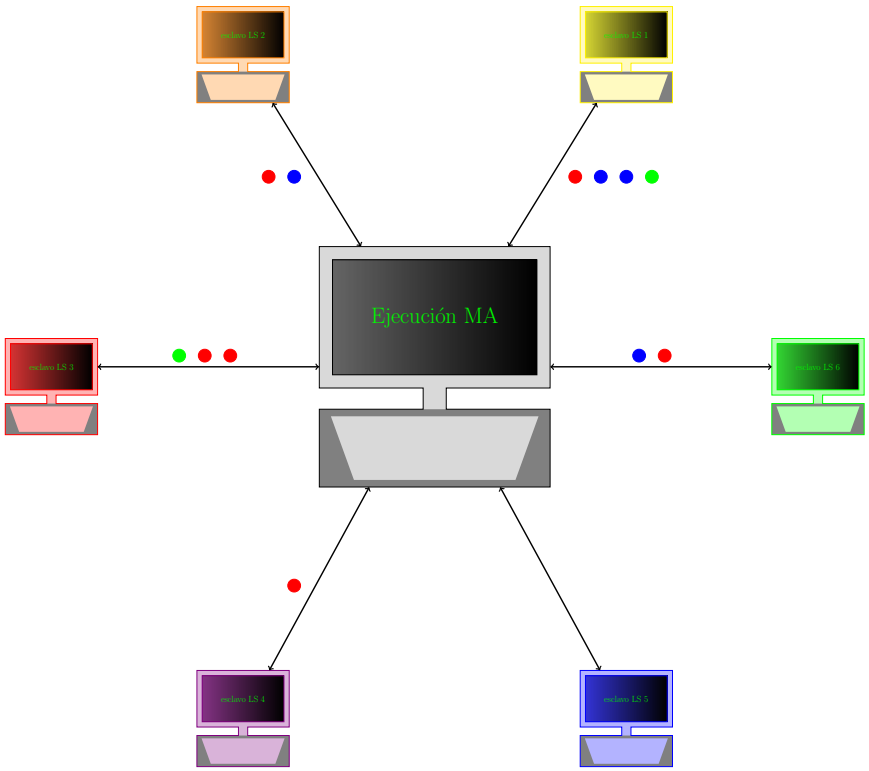


Figura 2.9: Esquema de un modelo maestro-esclavo donde el maestro ejecuta un MA que va enviando individuos a los esclavos para que estos ejecuten la operación de búsqueda local sobre los mismos.

lizado cuando el cuello de botella del algoritmo viene dado por el cálculo de la función objetivo del EA. En este caso, el servidor es responsable de las tareas de manipulación de la población distribuyendo los individuos entre los nodos para su evaluación para que finalmente los nodos los devuelvan ya evaluados al servidor central. La principal ventaja de este modelo es su capacidad para evaluar la población en paralelo.

- *Evaluación distribuida de las soluciones.* En este modelo la evaluación de cada solución se ejecuta en paralelo, es decir, se divide el cálculo de la función objetivo y se reparte entre diversos nodos que ejecutan un cálculo parcial y finalmente se recalcula la función objetivo completa a partir de dichos cálculos parciales. La ventaja de este modelo es que cuando la función objetivo es intensiva en uso de CPU y es paralelizable (lo cual no es siempre posible, ya que para ello dicha función se debe poder descomponer en varias funciones) se mitiga la penalización en rendimiento inducida por el cuello de botella producido por la función objetivo.

2.6.2.2. Modelo de Islas

Los computadores paralelos también se pueden utilizar para ejecutar diferentes simulaciones del mismo algoritmo en paralelo y de forma independiente unas de otras, sin que exista comunicación entre ellas. Cuando todas han finalizado o cuando alguna ha alcanzado un determinado valor o se ha cumplido algún criterio de finalización, se fusionan y se selecciona cuál es la mejor o mejores del conjunto completo de soluciones. Sin embargo, este modelo tiene algunas desventajas. Por ejemplo, si una de las ejecuciones alcanza una situación en la que se atasca en la búsqueda de soluciones porque haya llegado a un óptimo local, esta ejecución va a seguir en este estado durante el resto de la ejecución del algoritmo, aunque otras ejecuciones hayan alcanzado regiones del espacio de búsqueda con mejores soluciones. Lo ideal en estos casos sería que hubiera una comunicación entre las distintas ejecuciones, de forma que si la ejecución atascada en el óptimo local tuviera la posibilidad de recibir nuevas soluciones esta pudiera desplazarse a otra región del espacio de búsqueda más prometedora.

El desarrollo de metaheurísticas basadas en población –como por ejemplo los EAs– sobre arquitecturas paralelas ha sido un foco de interés desde la década de 1980 encontrándose en la literatura algunos antecedentes a lo que se conoce como el modelo de islas [160, 300, 311]. Actualmente la mayoría de las técnicas de búsqueda basadas en población utilizan algún tipo de disposición espacial de los individuos. En los modelos de islas, también denominados EAs distribuidos, modelos de granularidad gruesa[90] o modelos multi-deme[91], la población total se particiona en conjuntos más pequeños de poblaciones que se ejecutan en procesadores diferentes [52]. A menudo se suele utilizar el término isla para hacer referencia a una subpoblación o nodo, de manera que el conjunto de todas ellas forman el modelo de islas completo. Las islas funcionan de forma independiente la mayor parte del tiempo, como si fuesen ejecuciones aisladas del algoritmo, hasta que en un determinado momento bien de forma síncrona o asíncrona se produce una comunicación entre unas islas y otras transfiriendo soluciones entre ellas mediante un proceso denominado *migración*. Al igual que sucede en la evolución biológica, cuando poblaciones de una misma especie se aíslan unas de otras tiene lugar una evolución diferenciada, de forma que cada subpoblación se adapta a entornos diferentes y por lo tanto adquiere características específicas de dicho entorno. Así, los EAs que implementan los modelos de islas emplean aislamiento espacial para que cada isla explore regiones diferentes del espacio de búsqueda evolucionando de forma separada e incrementando la diversidad total y la probabilidad de escapar de óptimos locales. Todo esto unido a la permeabilidad entre subpoblaciones a través de los procesos migratorios definidos en estos modelos (tomados de las migraciones reales existentes en el mundo biológico) permite que el modelo de islas sea capaz de explorar regiones del espacio de búsqueda mucho más extensas que sus equivalentes secuenciales. Sin embargo, las

[90] *coarse-grained model*
[91] *multi-deme model*

migraciones tienen también ciertos riesgos, como por ejemplo, el hecho de que la inundación de las islas receptoras por soluciones de calidad puede ocasionar su rápida propagación acabando con la diversidad.

Relacionado con esto último se define el tiempo de absorción[92] como el tiempo necesario para que toda la población esté formada por la mejor solución [11]. Este tiempo se ha estudiado en poblaciones panmícticas y para diferentes esquemas de selección [315, 316] y también en modelos con poblaciones estructuradas en islas [11, 231, 317]. En general el tiempo de absorción debe ser mayor en algoritmos con múltiples subpoblaciones, ya que la diversidad de los individuos es mayor al evolucionar cada subpoblación de forma independiente y debido a que una solución únicamente puede expandirse por el resto de subpoblaciones a través de las migraciones. Evidentemente cuanto mayor sea la frecuencia de las migraciones más se parecerá el tiempo de absorción de un EA distribuido con varias subpoblaciones al de un EA secuencial.

La idea fundamental del modelo de islas es tener una *topología de migración*, es decir, un grafo con las islas como nodos y ejes conectando dos islas. Cada cierto tiempo los individuos seleccionados de cada isla se envían a las islas vecinas, es decir, aquellas que están conectadas por un eje en el grafo. Estos individuos se denominan *migrantes* y se incluyen en la isla destino después del correspondiente proceso de selección. Esta es una forma interesante de introducir tanto comunicación como competición entre islas. Además, las islas que se han quedado en un óptimo local sin poder seguir progresando tienen la oportunidad de recibir nuevos individuos procedentes de otras islas introduciendo nuevo material genético y por lo tanto, incrementando la diversidad y la probabilidad de salir de dicho óptimo local. En la Figura 2.10 se muestra un ejemplo de un modelo de islas básico con topología de anillo unidireccional.

Existen múltiples decisiones de diseño que afectan al comportamiento del modelo de islas –y a los EAs en general [110]–, algunas de las cuales se han tratado desde el punto de vista de las propiedades self-$\star$ (ver Sección 2.5) como los intervalos de migración [13, 53, 329], cuya adaptación se ha estudiado profusamente creando un modelo matemático que permite variarlos en función de resultados teóricos previos relacionados con la convergencia del algoritmo [288] o en función de las mejoras que se van produciendo (o no) en la calidad de las soluciones encontradas [232]. Otra cuestión adicional que se puede considerar es si todas las islas ejecutan el mismo algoritmo con idéntica parametrización, en cuyo caso se habla de *modelo homogéneo de islas* y en caso contrario de *modelo heterogéneo de islas*. A continuación se repasan algunos de los parámetros característicos del modelo de islas:

- *Política de emigración*: Los migrantes que se envían desde la isla emisora pueden ser eliminados de esta o no. En este último caso se enviarían copias de dichos migrantes y el mecanismo recibe habitualmente el nombre de *polinización*. Otros aspectos a tener en cuenta son los relativos a la selección

[92] *takeover*

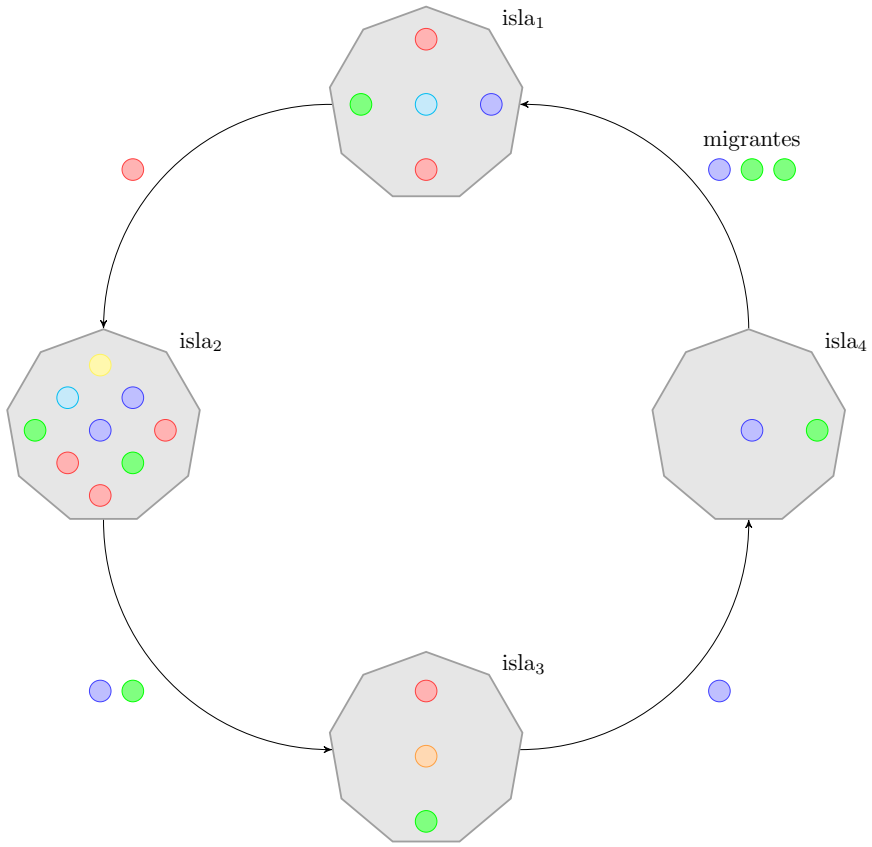


Figura 2.10: Ejemplo de modelo de islas con topología de anillo unidireccional y 4 islas.

de los migrantes, como por ejemplo, si se seleccionan los mejores, los peores o de forma aleatoria, por citar algunos ejemplos.

- *Política de inmigración*: Otra cuestión importante es la forma en la que los migrantes reemplazan a los individuos existentes en la isla receptora. Existen diversas posibilidades: se pueden reemplazar los peores individuos, hacerlo de forma aleatoria o utilizar el mismo tipo de selección utilizada por la isla para seleccionar a los padres. Hay otras posibilidades como reemplazar a los más similares o a los más diferentes o incluso decidir si se realiza una recombinación con individuos ya existentes en la isla destino.

- *Intervalo de migración*: El intervalo de tiempo entre migraciones determina la rapidez con la que se produce la transferencia de información entre islas. En ocasiones también se denomina *frecuencia de migración*, cuyo significado es recíproco. Si las migraciones son cada poco tiempo se extiende rápidamente la información de unas islas a otras; en cambio, migraciones ocasionales conducen a caminos con mayor exploración del espacio de búsqueda. Es interesante remarcar que si el intervalo de migración es ∞ el modelo de islas equivaldría a ejecuciones independientes. En vez de usar una migración

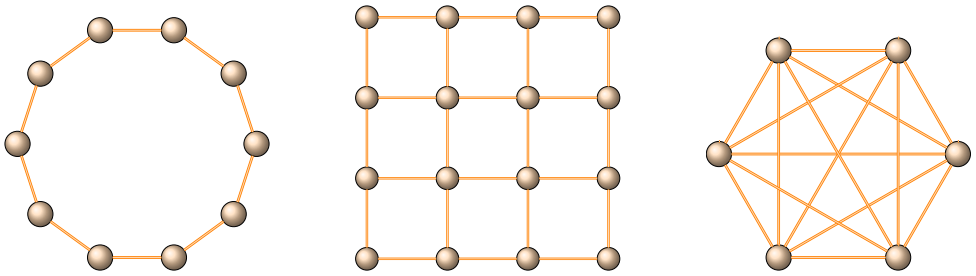


Figura 2.11: Ejemplos de topologías habituales. (a) Anillo Bidireccional. (b) Rejilla. (c) Grafo Completo. En el caso del ejemplo de la rejilla, si los extremos se conectan entre ellos se trataría de una rejilla toroidal.

periódica también es posible utilizar un mecanismo probabilístico de migración, es decir, cada isla determina de forma independiente la probabilidad de que sus emigrantes se envíen o no. Una probabilidad de migración de $1/\tau$ tiene un efecto similar que usar un intervalo de migración τ.

- *Número de migrantes*: El número de migrantes es otro factor que hay que tener en consideración cuando se diseña el algoritmo, ya que esto va a determinar lo rápido que una isla puede ser tomada por los individuos inmigrantes.

- *Topología de migración*: La topología del modelo de islas tiene un impacto importante sobre el comportamiento de la búsqueda, ya que determina los flujos de migración. La topología se representa mediante grafos dirigidos o no dirigidos –los grafos no dirigidos son un caso especial de los dirigidos. Las topologías más comunes son los anillos unidireccionales o bidireccionales, rejillas, hipercubos, grafos de escala libre, grafos completos, etc. La Figura 2.11 muestra algunos ejemplos de estas topologías.

- *Sincronismo*. Si el modelo se implementa de forma síncrona, cada población en ejecución en un nodo de computación debe esperar por otras poblaciones para proceder al proceso migratorio, en cambio en el modelo asíncrono el algoritmo continúa su ejecución, comprobando periódicamente si se han recibido nuevos individuos. La implementación asíncrona se recomienda cuando existen tiempos de ejecución muy diferentes entre unos nodos y otros. En cualquier caso no es necesario un servidor central para coordinar este proceso de recogida y envío de individuos entre poblaciones.

Desde un punto de vista estrictamente vinculado al rendimiento y la eficiencia se han analizado diversos aspectos de un EA [52, 110] concluyendo que la comunicación puede ser determinante [219]. La elección de la topología de migración y el intervalo de migración son decisiones críticas para obtener un buen rendimiento. En relación con la forma de medir los GAs paralelos hay modelos que predicen la

escalabilidad del algoritmo [54], calculando cuál es el número óptimo de procesadores que se pueden utilizar para minimizar el tiempo de ejecución, suponiendo un modelo en el que las migraciones tienen lugar cuando las subpoblaciones han convergido (o en cada generación, aunque este no es el caso más habitual, en muchas aplicaciones y modelos paralelos donde las migraciones ocurren cada cierto número de generaciones). La conclusión es que el número óptimo de procesadores n que minimiza el tiempo de ejecución es directamente proporcional a la raíz cuadrada del tamaño de la población μ y el tiempo de evaluación de la función de fitness T_f, e inversamente proporcional al tiempo medio de comunicación con otros procesadores T_c:

$$n = \mathcal{O}\left(\sqrt{\frac{\mu T_f}{T_c}}\right) \tag{2.6}$$

Esta ecuación establece formalmente que muchas aplicaciones prácticas pueden beneficiarse del uso de múltiples procesadores sobre todo conforme aumenta la complejidad de la función de fitness (aunque no siempre que se incrementa el número de procesadores se obtiene una eficiencia real mayor en la ejecución del algoritmo). Por otro lado, el análisis también muestra que distribuir los individuos disponibles entre múltiples islas sin que exista comunicación entre ellas no supone un beneficio considerable con respecto a las versiones secuenciales. Esto también sugiere que los modelos de islas con una topología de conexión con muchos vecinos son más efectivas que aquellas topologías con interconexiones dispersas. Siguiendo el mismo esquema se puede realizar el mismo análisis para la elección del grado óptimo de conexión de la topología que minimiza el coste de ejecución total del algoritmo [52].

2.6.2.3. EAs Celulares

Los EAs celulares proporcionan otro enfoque distinto (con una granularidad más fina de la paralelización) a la estructura espacial de la población. Al igual que en el modelo de islas existe una topología subyacente, siendo los anillos y los grafos toroidales de dos dimensiones las más habituales ya que se pueden implementar fácilmente sobre arquitecturas paralelas [235]. Cada individuo solo puede cruzarse con sus vecinos según la topología existente, lo que sucede normalmente en cada generación. De la misma manera que con el modelo de islas, el aislamiento espacial introduce, aunque de forma más gradual, la posibilidad de explorar diferentes regiones del espacio de búsqueda y aumentar la diversidad: una solución de calidad tardaría más en propagarse por toda la población dando tiempo a que se descubran otras soluciones de calidad en otras partes de la estructura y así promover la diversidad evitando que superindividuos acaben con ella rápidamente. Como ya se mencionó cuando se trató el modelo de islas, el tiempo de absorción también se ha estudiado en el contexto de los EA celulares [11, 314] y se han presentado resultados teóricos y empíricos sobre la presión selectiva tanto en EA celulares con topologías en anillo [147] como con topología toroidal [145].

Los EAs celulares implementan un mecanismo de selección y unas operaciones de variación descentralizadas, donde cada individuo tiene su propio conjunto de padres potenciales definido a través de una relación de vecindad. Para reemplazar los individuos se pueden seguir las mismas estrategias que fueron enumeradas para el modelo de islas. Las celdas se pueden actualizar de forma síncrona, tratándose en este caso de un EA celular síncrono, siendo la creación de una nueva población temporal la forma más habitual para implementarlo. Todos los padres se seleccionan a partir de la población actual y los nuevos individuos se van introduciendo en la población temporal. Al final del proceso la población actual se reemplaza por la población temporal. Alternativamente, las celdas se pueden actualizar de forma asíncrona, denominándose en este caso EA celular asíncrono. Se pueden definir diversas estrategias de actualización como en [10, 322]:

- *Elección uniforme*: La siguiente celda para actualizar se elige con probabilidad uniforme.
- *Barrido fijo por línea*: Las celdas se actualizan secuencialmente, concretamente línea a línea en una topología de rejilla o toroide.
- *Barrido fijo aleatorio*: Las celdas se actualizan secuencialmente, pero siguiendo un orden fijo. El orden se determina mediante una permutación de las celdas. Esta permutación se crea de forma aleatoria con probabilidad uniforme durante la inicialización y se mantiene durante toda la ejecución.
- *Barrido variable aleatorio*: Esta estrategia es similar a la de barrido fijo aleatorio, pero después de cada intercambio se crea una nueva permutación también de forma aleatoria.

En los EAs celulares se define una generación como el tiempo necesario para actualizar m celdas, siendo m el número de celdas de la rejilla. Las últimas tres estrategias garantizan que en cada generación cada celda se actualiza exactamente una vez. Esto produce un tratamiento de todas las celdas mucho más equilibrado. En cambio con el modelo de elección uniforme es probable que algunas celdas deban esperar un periodo de tiempo considerable antes de actualizarse. En el límite, el tiempo de espera para las actualizaciones sigue una distribución de Poisson.

2.6.2.4. Modelos Integrados

Se han propuesto otros modelos diferentes que bien integran varios de los modelos vistos hasta ahora o se definen en función de otras estructuras algebraicas. Por ejemplo, se ha estudiado un modelo unificado para estructuras de poblaciones basado en hipergrafos (Sprave [337]), es decir, una generalización de los grafos donde los ejes pueden conectarse con cualquier cantidad de vértices. El caso particular en el que cada hipereje contiene dos vértices diferentes se reduce a un

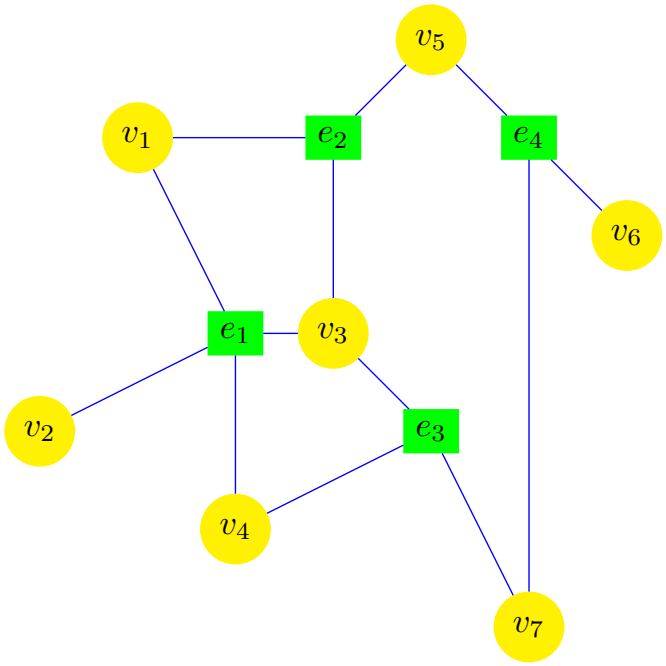


Figura 2.12: Ejemplo de hipergrafo $H = (V, E)$ con $V = \{v_1, \cdots, v_7\}$ y $E = \{e_1, e_2, e_3, e_4\} = \{\{v_1, v_2, v_3, v_4\}, \{v_1, v_3, v_5\}, \{v_3, v_4, v_7\}, \{v_5, v_6, v_7\}\}$.

grafo no dirigido. En el modelo de Sprave cada vértice representa un individuo y los hiperejes representan el conjunto de padres posibles para cada individuo (ver Figura 2.12). El modelo unifica varios modelos de población habituales:

- *Poblaciones panmícticas*: Dado un conjunto de vértices V hay un solo hipereje que contiene todo el conjunto de vértices. Esto significa que en una población panmíctica cada individuo tiene potencialmente a todos los individuos como padres.

- *Modelo de islas con migración*: Si se entiende la migración en el sentido de que los individuos que migran se eliminan, el conjunto de padres potenciales para un individuo dado comprende todos los inmigrantes potenciales y todos los individuos de su propia isla, excepto aquellos que han emigrado.

- *Modelo de islas con polinización*: Si se utiliza la polinización, el conjunto de padres potenciales comprende todos los inmigrantes y todos los individuos de su propia isla.

En el caso de modelos de granularidad gruesa, los hipergrafos pueden variar en el tiempo. Más concretamente, existen diferentes conjuntos de padres potenciales cuando se lleva a cabo la migración, en comparación con las generaciones en las que no hay migración. Esto puede tratarse considerando una estructura dinámica de población, es decir, en vez de considerar un hipergrafo fijo, se considera una

secuencia de hipergrafos variables durante todo el tiempo que dura la ejecución del algoritmo.

También es posible combinar algunas de las aproximaciones anteriores creando un modelo híbrido con dos niveles de paralelización [12]. Por ejemplo, se puede realizar un modelo de islas en el que cada isla ejecute un EA celular para promover la diversidad como en [98], donde se utiliza este modelo híbrido para resolver un problema de rutas de vehículos (estudiado también con algoritmos genéticos celulares [6, 8, 9]). Dicho modelo consiste en un GA paralelo en el que la población se estructura en dos niveles y de dos formas diferentes. En el primer nivel la población se descentraliza a través de un modelo de islas con topología de anillo unidireccional por lo que las migraciones solo ocurren entre los vecinos más próximos. En el segundo nivel cada isla es un GA celular con topología de rejilla toroidal 2D que implementa a su vez un modelo maestro-esclavo en el que el GA celular actúa como maestro enviando a los esclavos la ejecución de las operaciones de búsqueda local. Además de esta, existen otras implementaciones posibles como crear un modelo de islas jerárquico donde las islas son modelos de islas en sí mismos [17]. En este último caso, tendría sentido que el modelo de islas interno tenga una frecuencia de migración mayor que el modelo externo. Por otro lado, los modelos de islas y los EAs celulares se pueden implementar como modelos maestro-esclavo para conseguir un mejor rendimiento.

2.6.2.5. Plataformas para los Algoritmos Evolutivos Distribuidos

Los EAs habitualmente necesitan grandes cantidades de recursos computacionales cuando intentan abordar la resolución de un problema del mundo real. Estos requisitos computacionales vienen impuestos por varios motivos: las funciones de fitness son cada vez más complejas, el tamaño de las poblaciones suele ser elevado y el número de generaciones (iteraciones) necesarias para que el algoritmo converja a una solución de calidad es también grande. Como se ha expuesto con anterioridad es esto lo que hace necesario el uso de computación paralela. Las dos plataformas de recursos en paralelo más utilizadas en los últimos tiempos son las de computación voluntaria y las P2P.

Las plataformas de computación voluntaria tienen una muy buena relación coste/beneficio para ejecutar EAs sobre problemas complejos [66, 212], ya que permiten a los investigadores explotar los recursos de computación sin utilizar por los computadores. Esto ha hecho que cada vez más problemas que requieren elevadas cargas de cómputo se resuelvan de esta manera. Algunos ejemplos los tenemos con los modelos de predicción del clima [14], la física de alta energía [327] o la astrofísica [20, 327]. Este tipo de proyectos confían en los "voluntarios", haciendo a los usuarios partícipes del éxito final del proyecto. Con esta tecnología existen básicamente dos aproximaciones diferentes para los EAs paralelos:

- *Evaluación paralela del fitness*: se distribuyen los individuos que se van a evaluar simultáneamente entre varios computadores voluntarios. Esto es útil

fundamentalmente cuando la función objetivo consume una gran parte de los recursos computacionales.

- *Ejecución en paralelo de los experimentos*: esta aproximación es interesante cuando se quieren obtener resultados estadísticamente significativos por lo que es necesario realizar múltiples ejecuciones del mismo experimento. En este caso, las diferentes ejecuciones se distribuyen entre un número de computadores. También es útil cuando se requieren múltiples ejecuciones con diferentes parámetros para encontrar la configuración óptima del algoritmo.

Por otro lado las plataformas P2P son también una buena opción sobre la que implementar EAs. Al igual que cualquier EA distribuido, un EA P2P tiene dos facetas claramente diferenciadas: por un lado aquella relacionada con los cambios estructurales a los que hay que someter al algoritmo para desplegarlo sobre varios procesadores y por otro, lo relacionado con las expectativas en cuanto al rendimiento del algoritmo. Además, los EAs distribuidos son una forma de preservar la diversidad genética a la vez que permiten mejorar los tiempos de ejecución del EA [215–217]. En general, los EAs paralelos utilizan tres modelos fundamentalmente: maestro-esclavo, islas y EAs celulares. Sin embargo, todos ellos no encajan con los requisitos propios de las plataformas P2P, tales como descentralización, gran escalabilidad o tolerancia a fallos. De entre todos, es el modelo de islas el que cumple todos los requisitos que debe tener un algoritmo ejecutado sobre una plataforma P2P, ya que cada isla se puede ejecutar sobre uno de los computadores que forman parte de la red P2P (descentralización), se pueden añadir nuevas islas o eliminar algunas de las ya existentes (escalabilidad) y permiten introducir técnicas de tolerancia a fallos.

En los últimos años se han realizado numerosos trabajos estudiando el comportamiento y el rendimiento de los EAs sobre plataformas distribuidas y se ha analizado cómo se puede incrementar su potencia de cómputo [157]. Con respecto a los EAs basados en islas sobre este tipo de plataformas computacionales, la volatilidad de los nodos puede provocar que se pierda la mejor solución (si la isla que la contiene desaparece del sistema completo antes de tener la oportunidad de migrar a otra isla) [172] y en general, esto afectará a la diversidad de la población. Esto se podría ajustar tomando medidas correctivas, e.g., usar estrategias de tolerancia a fallos para recuperarse de ellos [212, 214, 227, 228, 321] o medidas preventivas, en las que el algoritmo sea capaz de auto-adaptarse a los fallos conforme estos suceden como se vio en la Sección 2.5, intentando mantener una variedad genética –y memética en el caso de un MA– durante todo el tiempo de ejecución. Lo segundo puede suponer una ventaja adicional, ya que al tratarse de un procedimiento autónomo y descentralizado no es necesario conocer el estado global del sistema.

Con el auge de la computación en la nube durante la primera mitad de la década de 2010 y el desarrollo de servicios para este tipo de entornos se han implementado también EAs distribuidos basados en este modelo [241, 244], los

cuales hacen uso de un servicio de almacenamiento síncrono para intercambiar información entre los individuos de las poblaciones. Durante el proceso el algoritmo puede compartir individuos para mejorar el proceso de búsqueda –incrementando la diversidad. Estos individuos se colocan en una zona común y permanecen en ella hasta que se añaden a alguna población (a una o a varias). Siguiendo en esta línea, las arquitecturas orientadas a servicios se han empleado para diseñar e implementar EAs distribuidos [136, 137]. Una de las grandes ventajas de estas arquitecturas es que son independientes del lenguaje de programación utilizado, ya que su funcionamiento está basado en la oferta de servicios de forma que los clientes únicamente tienen que hacer uso de ellos sin necesidad de conocer cómo se han desarrollado ni con qué tecnología.

Existen otras plataformas implementadas en la nube como *EvoSpace* [138–140]. Se trata de un servicio en la nube para el desarrollo de EAs distribuidos basado en un modelo espacial de memoria compartida entre varios procesadores. Los clientes remotos se denominan *EvoWorkers*, los cuales se inicializan con una pequeña población seleccionada de forma aleatoria que se utiliza como punto de partida de un EA local que se ejecuta en el computador cliente. De forma periódica se conectan a *EvoSpace* para obtener un conjunto de individuos de la población global, realizar las correspondientes operaciones evolutivas sobre ellos y finalmente devolver un conjunto de nuevos individuos a la población global. Los *EvoWorkers* realizan el proceso de búsqueda en paralelo y asíncronamente interaccionan entre ellos a través del repositorio central. *EvoSpace* se ha diseñado para poder utilizar diferentes tipos de EAs.

Capítulo 3

Análisis de Algoritmos Meméticos con Auto-Generación

Los MAs auto-generados [200, 203] generan sus propios operadores de búsqueda local haciendo que estos evolucionen junto con el contenido cromosómico de los individuos. Se puede por lo tanto considerar que existen dos niveles de evolución: por un lado la evolución genética de los EA clásicos y por otro una evolución memética. Más concretamente, en un MMA la representación de los individuos se ve aumentada con material memético, mediante el cual se determina el operador de búsqueda local que se va a utilizar para cada individuo concreto. Además, el material memético se hereda de padres a hijos a través del mecanismo de recombinación e incluso puede introducirse un procedimiento de mutación memética con una probabilidad determinada replicando de esta manera parte del ciclo evolutivo genético habitual. En resumen, los MMAs son MAs que explícitamente hacen evolucionar los memes, es decir, expresiones no genéticas de estrategias de resolución de problemas.

Por otra parte, aunque los algoritmos basados en poblaciones frecuentemente se han usado a través de una aproximación panmíctica (por medio de la cual cualesquiera dos soluciones dentro de la población podrían interactuar con propósitos reproductivos), se han propuesto estructuras de población más generales en las últimas décadas (ver e.g. [67, 160]). Sin embargo, el desarrollo de tales estructuras sobre los MMAs se ha explorado en este sentido con menor dedicación.

En esta tesis doctoral se ha considerado una clase de MMA con una estructura espacial bien definida (ver Sección 3.1) en la que los memes se definen como reglas de reescritura cuya longitud puede ser fija durante la ejecución del algoritmo (ver Sección 3.2) o variable, auto-adaptándose en este último caso durante el proceso de búsqueda (ver Sección 3.3), estando sujetos en ambos casos a un proceso evolutivo similar al de la evolución genética. Finalmente se ha considerado una aproximación alternativa al uso de los operadores genéticos tradicionales basada en el uso de EDAs, creando de esta forma un modelo híbrido de MMA con un potente motor probabilístico capaz de encontrar las relaciones existentes entre los individuos, tanto a nivel genético como memético (ver Sección 3.4).

3.1. Algoritmos Multimeméticos

Los primeros MAs eran simplemente vistos como algoritmos híbridos, un proceso simbiótico entre los algoritmos de búsqueda global basados en poblaciones –a menudo tomando la forma de un EA– y una etapa de evolución cultural –habitualmente a través de un proceso de búsqueda local. Posteriormente se desarrolló el concepto de los algoritmos multi-meme [200, 206], en los cuales el material memético se codifica como parte del genotipo incorporando los principios de transmisión y selección meméticas en su diseño. A continuación se muestra la estructura y funcionamiento de un algoritmo de estas características considerado en este trabajo.

3.1.1. Estructura de un MMA

La idea principal de los MMAs es el tratamiento explícito de los memes dentro del proceso evolutivo. Además del uso de memes incrustados dentro de los individuos, el MMA que se va a considerar durante toda esta tesis doctoral sigue el marco de trabajo definido por Smith [333, 334]. Con más detalle, se considera que cada individuo de la población transporta un genotipo binario y un único meme, que representa una regla de reescritura. Por otro lado se trata de un MMA en el que los padres se seleccionan usando torneo binario y se utiliza la recombinación, la mutación y la búsqueda local (a través del meme que transporta cada individuo como se ilustró con anterioridad) para generar la población de descendientes, la cual reemplaza a los peores padres (ver Algoritmo 2). La fase de búsqueda local consiste en la aplicación del meme (a través de la función AplicarMeme) al genotipo correspondiente utilizando para ello la regla de reescritura que representa a cada meme según se explica en la Sección 3.1.2.

Considérese que existen dos funciones diferentes de cruce (CruzarGenes y CruzarMemes) y mutación (MutarGenes y MutarMemes). Una de ellas se aplica a los genotipos y la otra a los memes, lo que permite establecer el proceso evolutivo tanto en la estructura genética del individuo como en la memética. El resto de funciones del algoritmo son las habituales de los EAs: InicializarPoblación para realizar una inicialización aleatoria de la población, tanto desde el punto de vista genético como memético (se inicializan también las reglas de reescritura de forma aleatoria), EvaluarPoblación para aplicar la función objetivo considerada al genotipo de cada individuo, SeleccionarPadres para seleccionar los padres que van a ser los progenitores de la descendencia, y finalmente ActualizarPoblación para proceder al reemplazo de los individuos de la población por los nuevos individuos generados.

En las siguientes secciones se describe en primer lugar la representación de estos memes para posteriormente definir las estructuras espaciales que se han considerado durante el desarrollo de este trabajo.

Algoritmo 2: Pseudocódigo de un MMA

```
function MMA (↓ I, ↓ par) returns S
// I: Instancia del problema
// par: Parámetros del problema
// S: Solución devuelta
pop ← InicializarPoblación(I, par);
pop ← EvaluarPoblación(I, par, pop);
while ¬ CondiciónFin do
    // Selección aleatoria
    pop' ← SeleccionarPadres(I, par, pop);
    // Cruce genético
    pop' ← CruzarGenes(I, par, pop');
    // Mutación genética
    pop' ← MutarGenes(I, par, pop');
    // Cruce memético
    pop' ← CruzarMemes(I, par, pop');
    // Mutación memética
    pop' ← MutarMemes(I, par, pop');
    // Mejora local → Aplicación del meme a cada descendiente
    for i ∈ [1 ··· par.λ] do
        pop'(i) ← AplicarMeme(I, par, pop'(i));
    end
    pop' ← EvaluarPoblación(I, par, pop');
    pop ← ActualizarPoblación(I, par, pop, pop');
end
return Mejor(pop)
```

3.1.2. Representación de Memes

Los memes son expresiones no genéticas de estrategias de resolución de problemas y por definición se pueden representar de muchas formas diferentes, dependiendo del nivel de abstracción deseado y de la dependencia con el problema considerado. Siguiendo algunas ideas postuladas por Smith [334] en el contexto de la optimización de funciones pseudobooleanas, se considera en este capítulo que los memes se expresan como reglas de reescritura basadas en patrones [*condición*→*acción*] de la siguiente forma:

Definición 38. *Sea* $[C \to A]$ *un meme, donde* $C, A \in \Sigma^r$ *con* $\Sigma = \{0, 1, \#\}$ *y* $r \in \mathbb{N}$.

En este alfabeto ternario '#' representa un símbolo comodín que se interpreta de la siguiente manera:

- En el antecedente de la regla denota un símbolo que se ignora, es decir, que coincide tanto con '0' como con '1'.

- En el consecuente de la regla denota un símbolo transparente, significando que la correspondiente posición del genotipo permanece inalterada.

Definición 39. *Sea $g_1 \cdots g_n$ un genotipo. Un meme $[C \to A]$, con $C = c_1 \cdots c_r$ se podría aplicar sobre cualquier posición $i \in \{1, \cdots, n-r+1\}$ del genotipo para la que se cumpla que $g_i \cdots g_{i+r-1} = c_1 \cdots c_r$.*

Para el propósito de esta comparación, los símbolos comodín en el antecedente se emparejan con cualquier símbolo del genotipo.

Definición 40. *Sea i una posición del genotipo sobre la que se puede aplicar el meme, entonces su consecuente $A = a_1 \cdots a_r$ debe ser implantado en esa misma porción del genotipo, es decir, $g_i \cdots g_{i+r-1} \leftarrow a_1 \cdots a_r$.*

En este caso, la interpretación de los símbolos comodín en el consecuente es la de símbolos transparentes, esto es, se mantienen sin cambiar los símbolos correspondientes en el genotipo. Esto último supone una diferencia con respecto a la propuesta de Smith [334] en la que los comodines en el consecuente representan el complemento binario del gen original.

Por ejemplo, sea un genotipo 11101100, y sea una regla 10# $\to$ 0#1. Una posible aplicación de la regla podría ser la siguiente:

$$11\overbrace{101}^{C}100 \xrightarrow{\text{regla}} 11\underbrace{001}_{A}100$$

En este ejemplo hay otro punto potencial de aplicación de la regla, concretamente en la sexta posición, produciendo un genotipo 11101001. Para mantener el coste total del proceso bajo control se define un parámetro adicional w para determinar el número máximo de aplicaciones de reglas por individuo. El mejor vecino que se ha generado mediante el proceso anterior se mantiene en la población si es mejor que el genotipo actual (ver Algoritmo 3 para más detalles). Para evitar sesgos por posición, el orden en el que se escanea el genotipo para detectar sitios donde potencialmente se podría aplicar el meme se aleatoriza. La función BuscarPos encuentra la coincidencia i-ésima del antecedente de la regla en el genotipo siguiendo el orden de bits establecido por una permutación aleatoria para hacer que la búsqueda no esté sesgada y que no siempre se recorra el genotipo de la misma manera. Es interesante resaltar que el Algoritmo 3 se podría aplicar iterativamente sobre un mismo individuo, es decir, cada vez que se aplica un meme a un individuo se le puede volver a aplicar al individuo resultante. Este proceso podría considerarse como una búsqueda en profundidad dentro del proceso de búsqueda local. Por simplicidad y por no ser determinante para los propósitos de esta tesis doctoral, en los MMAs desarrollados siempre se ha considerado que la profundidad de la aplicación de los memes es la unidad. Finalmente, se puede considerar que gracias a la definición de patrones utilizada, el proceso completo de reescritura del genotipo por medio de reglas puede verse como una forma de

Algoritmo 3: Mejora Local

function AplicarMeme ($\downarrow I, \downarrow par, \downarrow \langle g, m\rangle$) **returns** $\langle g', m'\rangle$
// I: Instancia del problema
// par: Parámetros del problema
// $\langle g, m\rangle$: Individuo $\langle genotipo, meme\rangle$, donde meme tiene la forma $[C \to A]$
// $\langle g', m'\rangle$: Individuo resultante $\langle genotipo, meme\rangle$, donde meme tiene la forma $[C \to A]$
$\langle g', m'\rangle \leftarrow \langle g, m\rangle$;
// Se genera una permutación de los bits del genotipo
$p = \text{RandPerm}(\text{Longitud}(g))$;
for $i \in [1 \cdots par.w]$ **do**
  // Se busca en el genotipo una posición para la que coincida el antecedente de la regla
  $\langle \rho, OK\rangle \leftarrow \text{BuscarPos}(\langle g, m\rangle, p, i)$;
  $\langle g'', m''\rangle \leftarrow \langle g, m\rangle$;
  if OK **then**
    $g''_\rho \cdots g''_{\rho+r-1} \leftarrow a_1 \cdots a_r$;
    $\text{Evaluar}(g'', par)$;
    if $\langle g'', m''\rangle \succ \langle g', m'\rangle$ **then**
      $\langle g', m'\rangle \leftarrow \langle g'', m''\rangle$;
    end
  end
end
return $\langle g', m'\rangle$

heurística para manipular esquemas, inyectando instancias de nuevos esquemas en lugar de los existentes cuando la función de fitness determina que el nuevo es mejor. La principal ventaja de tener memes enlazados a los individuos es que dota al algoritmo de la habilidad de descubrir las vecinos más apropiados para una solución determinada, así como una efectividad mayor en la exploración de dichos vecinos.

3.1.3. Estructura Espacial

La estructura espacial de la población está relacionada con la estructura topológica sobre la que los individuos de la población se proyectan. Más concretamente, sea T esta estructura topológica compuesta por μ posiciones (μ coincide con el tamaño de la población), cada una de ellas identificada por un índice $i \in \{1, \cdots, \mu\}$. Se caracteriza esta estructura utilizando una matriz lógica S de tamaño $\mu \times \mu$. Cada entrada de S_{ij} representa una interacción potencial entre dos posiciones de la estructura. Para ser exactos, sea S_{ij}= `true`, si, y solo si, el

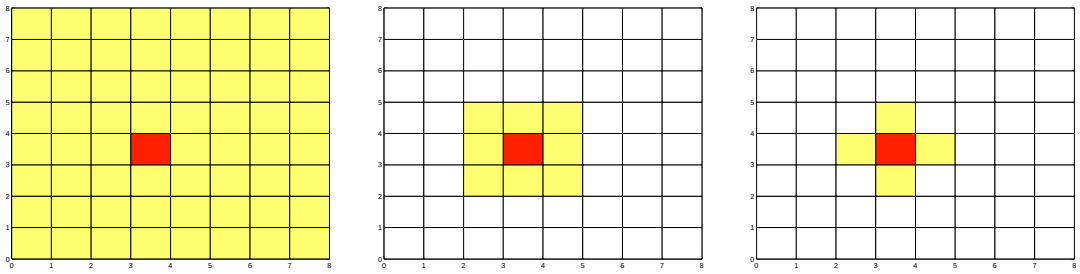


Figura 3.1: Ilustración de los diferentes tipos de vecindarios considerados. Las celdas rojas representan un individuo arbitrario y las celdas amarillas denotan el conjunto de celdas vecinas del individuo. Los vecindarios son de izquierda a derecha: panmíctico, Moore y von Neumann. En los dos últimos casos, $\rho = 1$.

individuo de la posición i-ésima puede interactuar con el individuo de la posición j-ésima.

En esta sección se han considerado matrices de interacción inducidas por una disposición espacial particular de las posiciones de los individuos distribuidas en una rejilla: sea $\mu = a \times b$; cada posición i puede entonces representarse por un par de coordenadas $(i_x, i_y) \in \{1, \cdots, a\} \times \{1, \cdots, b\}$. Ahora, sea $d : (\{1, \cdots, a\} \times \{1, \cdots, b\})^2 \to \mathbb{N}$ una medida de la distancia entre localizaciones. Dado un cierto vecindario de radio ρ, se toma $S_{ij} \Leftrightarrow (d(i, j) \leqslant \rho)$, es decir, dos localizaciones pueden interactuar si están dentro de un cierto umbral de distancia. Se presentan diversas estructuras espaciales y diversas alternativas de medidas de distancia. Algunas de las más habituales se enumeran a continuación:

1. Panmixia: $d(\cdot, \cdot) = 0$.
2. vecindad de Moore: $d((i_x, i_y), (j_x, j_y)) = \text{máx}(|i_x - j_x|, |i_y - j_y|)$.
3. vecindad de von Neumann: $d((i_x, i_y), (j_x, j_y)) = |i_x - j_x| + |i_y - j_y|$.

Las operaciones sobre distancias anteriores se realizan módulo el rango de coordenadas para hacerlas de forma toroidal. La Figura 3.1 ilustra estas estructuras espaciales.

3.2. Análisis de la Propagación de Memes

Una cuestión sumamente interesante que surge en el contexto de los MMAs es cómo se propagan y difunden los memes a través de la población. A pesar de que la dinámica de la población se ha estudiado bastante a fondo en el caso de los EAs –e.g., [11, 145, 189, 318, 319]–, el escenario es considerablemente más complejo cuando hay memes involucrados: a diferencia de los genotipos (que corresponden a soluciones y por lo tanto se pueden evaluar de acuerdo con el problema bajo consideración), los memes solo se pueden valorar de forma indirecta a través del

efecto que tienen sobre los genotipos. Además, los memes evolucionan en los MMAs junto con las soluciones uniéndose a ellas. Dado que este acoplamiento es parte del proceso auto-adaptativo, el algoritmo debe ser capaz de descubrir las relaciones gen-meme que mejor congenian entre los individuos de la población, y esto debe ser capaz de hacerlo únicamente a partir de la información sobre la calidad de las soluciones genéticas (i.e., a partir de la información sobre el fitness). Esta sección estudia cómo se propagan los memes en el escenario descrito anteriormente, considerando un modelo selecto-lamarckiano idealizado de MMA conducido por la selección genética sobre una estructura espacial subyacente. Para finalizar, la sección se centra en un MMA real, analizando su comportamiento sobre diferentes problemas de test con el objetivo de corroborar las conclusiones teóricas.

3.2.1. Modelo Selecto-lamarckiano

Considérese un modelo abstracto de MMA en el que cada agente viene caracterizado por un par $\langle g, m\rangle \in D^2$, para algún $D \subset \mathbb{R}$. El primer miembro del par (g) representa el genotipo, el cual se puede equiparar por simplicidad al fitness. En cuanto al segundo miembro (m), representa un meme. Para ser más exactos, este valor captura el *potencial de mejora* de ese meme, es decir, es una medida de cómo de buenas pueden llegar a ser las soluciones aplicándoles el meme. Se supone que existe una función $f : D^2 \to D$ monótona creciente con respecto a su primer argumento que encapsula la aplicación del meme sobre el genotipo, es decir, el efecto de una sola etapa del aprendizaje lamarckiano. Por lo tanto, un agente $\langle g, m\rangle$ se convierte en $\langle f(g, m), m\rangle$ después de la aplicación del meme, donde:

$$\lim_{n\to\infty} f^n(g, m) = m \quad \text{if } g < m \tag{3.1}$$

$$f(g, m) = g \quad \text{if } g \geqslant m \tag{3.2}$$

Aquí $f^n(g, m)$ es la n-ésima composición de la función sobre el primer argumento, es decir, $f(f(\cdots f(f(g, m), m), \cdots, m), m)$. Intuitivamente estas condiciones indican que el fitness de una solución se puede mejorar cuando se aplica sobre él un meme de calidad, aproximándose asintóticamente al potencial del meme cuando se aplica muchas veces. Téngase en cuenta que mientras esto es una caracterización muy idealizada del potencial del meme (ya que en general este potencial no a va ser absoluto sino que puede depender de una relación más compleja entre el meme, el genotipo y la configuración del problema) sirve como un intento inicial para estudiar varias cuestiones relacionadas con la propagación de los memes por la población de agentes.

La población P del MMA es, por lo tanto, un conjunto de μ agentes, $P = [\langle g_1, m_1\rangle, \cdots, \langle g_\mu, m_\mu\rangle]$, dotados de una estructura espacial que restringe la comunicación entre los mismos. Esta estructura espacial viene caracterizada por una matriz lógica S de tamaño $\mu \times \mu$, donde S_{ij} es `true` si, y solo si, el

Algoritmo 4: Modelo Selecto-Lamarckiano

```
for i ∈ [1 ··· μ] do
    InicializarIndividuo⟨g_i, m_i⟩;
end
while ¬ Convergencia (P) do
    i ← URand(1, μ) // Se selecciona una posición aleatoriamente
    ⟨g, m⟩ ←Selección(P, S, i);
    g' ← f(g, m) // Mejora Local
    P ← Reemplazo(P, S, i, ⟨g', m⟩);
end
```

agente localizado en la posición i-ésima se puede comunicar con el agente localizado en la posición j-ésima (considérese que únicamente se coloca un agente en cada posición). Dado que en la subsección siguiente el interés se centra en observar la dinámica de propagación de los memes, se considera una extensión de un modelo de EAs de solo-selección (i.e., usando solo selección/reemplazo pero sin operadores de variación) en el que se añade una etapa de mejora local típica de los algoritmos meméticos. En el Algoritmo 4 se muestra un esquema del modelo descrito.

Después de una adecuada inicialización del contenido de la población, el algoritmo entra en un ciclo de selección más mejora hasta que la población de agentes converge. La convergencia aquí se considera desde una perspectiva memética, es decir, el algoritmo termina cuando la población está compuesta por un conjunto de memes homogéneo (independientemente de que haya todavía diversidad a nivel genético o no). Con respecto al funcionamiento interno del algoritmo, se asemeja bastante a la estrategia de actualización mediante barrido variable aleatorio (ver Sección 2.6.2.3).

3.2.2. Propagación de Memes

Una vez definido el modelo general en la sección previa, es posible considerar algunas características cualitativas de la propagación de los memes que se pueden extraer de él. Supóngase que la selección se realiza por torneo binario, es decir, una vez que se selecciona una posición i, se selecciona una posición vecina j de $\mathcal{N}(i) = \{j \mid S_{ij}\}$, y se conserva el agente con el mejor fitness. En relación con el reemplazo, supóngase que el agente mejorado reemplaza al agente que perdió en el torneo previo.

El interés se centra en analizar el número de copias de cada meme dentro del conjunto total de memes. Sea $N(m, g, t)$ el número de instancias del meme m unidas al genotipo g en el instante t, suponiendo por simplicidad que D es algún dominio discreto. Si se divide esta cantidad por el tamaño del conjunto de memes μ se obtiene $p(m, g, t)$, es decir, la fracción de la población compuesta por

el meme m unido al genotipo g en el instante t. En cada paso de iteración del sistema, el número de copias se puede estimar de la siguiente manera:

$$N(m, g, t+1) = N(m, g, t) + C(m, g, t) - D(m, g, t) \tag{3.3}$$

donde $C(m, g, t)$ y $D(m, g, t)$ representan respectivamente el número de copias esperado del meme m unidas al genotipo g que se crearon o destruyeron en el instante t. La creación de una nueva copia se puede conseguir mediante el efecto combinado de la selección de un agente apto con el meme m y la aplicación del meme al correspondiente genotipo. Esto se puede expresar de la siguiente forma:

$$C(m, g, t) = \sum_{g'} \sigma(m, g', t) p(g' \xrightarrow{m} g) \tag{3.4}$$

donde $\sigma(m, g', t)$ es la probabilidad de la selección de un agente que lleva el meme m y el genotipo g' en el instante t y $p(g' \xrightarrow{m} g)$ es la probabilidad de que la aplicación del meme m sobre el genotipo g' produzca el genotipo g. La primera cantidad se puede calcular como la probabilidad de que el torneo binario elija dos agentes con meme m y genotipo g o solo un agente con esta estructura pero con un fitness mejor que su competidor:

$$\sigma(m, g, t) = p(m, g, t)^2 + 2\left\{p(m, g, t)\left[1 - p(m, g, t)\right]\right\} \cdot \frac{\sum_{m'} \sum_{g' < g} p(m', g', t)}{1 - p(m, g, t)} \tag{3.5}$$

donde el último factor es la probabilidad de que el fitness del competidor sea peor que el fitness del individuo seleccionado (o lo que es lo mismo, la probabilidad de que el fitness del competidor sea peor que g, siempre y cuando no esté en el agente $\langle g, m \rangle$). Esta expresión supone que la distribución global de memes/genotipos sobre la población completa es la misma que la de los vecinos locales. Obviamente, esto se sostiene para el caso panmíctico en el que cualesquiera dos agentes son vecinos por lo que se puede suponer este caso inicialmente y considerarlo como una primera aproximación hacia situaciones más generales.

Con respecto a la destrucción de una copia de un par genotipo/meme determinado, esto puede surgir a través de la selección de un par y la siguiente aplicación de la mejora local (lo que alterará el genotipo) o a través del reemplazo por un agente con un fitness mejor. El primer caso también requiere que el otro agente seleccionado en el torneo sea una copia del mismo par, para que pueda ser sustituido posteriormente por el agente mejorado. Por lo tanto,

$$D(m, g, t) = \sum_{g' \neq g} p(m, g, t)^2 p(g \xrightarrow{m} g') + \tilde{\sigma}(m, g, t) \ . \tag{3.6}$$

donde la probabilidad de reeemplazo $\tilde{\sigma}(m, g, t)$ se puede expresar como:

$$\tilde{\sigma}(m,g,t) = 2\left\{p(m,g,t)\left[1-p(m,g,t)\right]\right\} \cdot \frac{\sum_{m'}\sum_{g'>g} p(m',g',t)}{1-p(m,g,t)} \tag{3.7}$$

Considérese ahora la evolución del sistema a corto y medio plazo, antes de que un meme determinado empiece a saturar la población. En esta situación los memes se extienden ampliamente sobre los genotipos, así que $p(m,g,t) \ll 1$, y tomando los términos cuadráticos $p(m,g,t)^2$ como aproximadamente 0 se tiene:

$$\sigma(m,g,t) = 2p(m,g,t)\sum_{m'}\sum_{g'<g} p(m',g',t) \tag{3.8}$$

$$\tilde{\sigma}(m,g,t) = 2p(m,g,t)\sum_{m'}\sum_{g'>g} p(m',g',t) \tag{3.9}$$

Sustituyendo de nuevo en las Ecuaciones (3.4) y (3.6) se obtiene:

$$C(m,g,t) = 2\sum_{g'} p(m,g',t)\sum_{m'}\sum_{g''<g'} p(m',g'',t)p(g' \xrightarrow{m} g) \tag{3.10}$$

$$D(m,g,t) = 2p(m,g,t)\sum_{m'}\sum_{g'>g} p(m',g',t) \tag{3.11}$$

Ya que $p(g' \xrightarrow{m} g) = 0$ para $g < g'$ o $m < g$, la Ecuación (3.10) se reduce a

$$C(m,g,t) = 2\sum_{g''<g'\leqslant g}\sum_{m'} p(m,g',t)p(m',g'',t)p(g' \xrightarrow{m} g) \tag{3.12}$$

Si $m \leqslant g$ entonces $p(g' \xrightarrow{m} g)$ es 1 si $g' = g$ y 0 en otro caso. Consecuentemente, la diferencia $\Delta_g^m(t) = C(m,g,t) - D(m,g,t)$ es en este caso:

$$\begin{aligned}\Delta_g^m(t) = {} & 2p(m,g,t)\sum_{m'}\sum_{g''<g} p(m',g'',t) - 2p(m,g,t)\sum_{m'}\sum_{g''>g} p(m',g'',t) \\ = {} & 2p(m,g,t)\cdot\sum_{m'}\left[\sum_{g''<g} p(m',g'',t) - \sum_{g''>g} p(m',g'',t)\right]\end{aligned} \tag{3.13}$$

Si se considera ahora el signo de la diferencia en la expresión anterior, fundamentalmente se obtiene que los memes *inertes* (i.e., los memes que no son capaces de seguir mejorando los individuos que los contienen) pueden progresar, uniéndose a aquellos agentes mejores que la mediana de la población.

Por otro lado, consideremos ahora el caso en el que $m > g$. En esta situación, $p(g' \xrightarrow{m} g)$ es 1 si $g' = f^{-1}(g,m)$ y 0 en otro caso, donde se denota por $f^{-1}(g,m)$ el valor del genotipo tal que $f(f^{-1}(g,m),m) = g$. Usando g^{-m} como notación

abreviada para $f^{-1}(g, m)$,

$$\begin{aligned}\Delta_g^m(t) = \quad & 2p(m, g^{-m}, t) \sum_{m'} \sum_{g'' < g^{-m}} p(m', g'', t) - 2p(m, g, t) \sum_{m'} \sum_{g' > g} p(m', g', t) \\ = \quad & \sum_{m'} \left[2p(m, g^{-m}, t) \sum_{g'' < g^{-m}} p(m', g'', t) - 2p(m, g, t) \sum_{g' > g} p(m', g', t) \right] .\end{aligned} \tag{3.14}$$

El signo de esta expresion depende del equilibrio entre la bondad de los genotipos en la cuenca de atracción de g y la propia bondad de g. Considérese que en general, estas cantidades están interrelacionadas en el sentido de que cuanto mejores son los genotipos en la cuenca de atracción de g, mejor puede esperarse que sea g. Esto no solo significa que los memes *activos* proliferen más y más cuando se unan a buenas soluciones como se podría esperar, sino que también esos memes con alto potencial pueden encontrar su camino hacia las etapas finales de la evolución siempre que tengan tiempo suficiente para mejorar los individuos donde se encuentran (conviene recalcar aquí que lo buenas que son las soluciones es algo que evoluciona con el tiempo como un efecto de la aplicación de los memes). Esto sugiere que los modelos con una convergencia genética más baja pueden tener un efecto beneficioso sobre la propagación de los memes buenos, permitiendo a estos últimos tener el tiempo suficiente para progresar en la población y vencer el efecto producido por memes de mala calidad que han ido progresando porque se han incorporado a genes buenos –comensalismo[1] en términos biológicos. En la próxima sección se proporciona un análisis más cuantitativo de este efecto mediante simulaciones numéricas.

3.2.3. Simulaciones Numéricas

La experimentación numérica aspira a explorar empíricamente la dinámica de la propagación de los memes y cómo le afectan otros factores tales como el tamaño de la población, el potencial relativo de mejora de los memes y la estructura espacial subyacente de la población. Con respecto al tamaño de la población, se han considerado los siguientes valores $\mu \in \{100, 256, 400, 625\}$. Estos valores cubren un amplio rango de tamaños de población diferentes y son además, cuadrados perfectos, lo que es importante para una de las estructuras espaciales consideradas, concretamente la rejilla cuadrada toroidal con vecindad de von Neumann. Se ha considerado que el radio de la vecindad es $\rho = 1$, lo que conduce al tradicional vecindario Norte-Sur-Este-Oeste (además de la posición actual). La otra estructura espacial que se ha considerado es el modelo panmíctico en el que todas las posiciones están conectadas. En cualquier caso, se ha considerado la

[1] *hitchhiking*

siguiente función:

$$f(g,m) = \begin{cases} g & \text{if } g \geqslant m \\ (g+m)/2 & \text{if } g < m \end{cases} \tag{3.15}$$

para representar la acción de los memes. Intuitivamente, esta función proporciona mejoras más pequeñas conforme va mejorando la calidad del genotipo, lo que sucede a menudo en la práctica. Todos los experimentos se han promediado sobre 100 ejecuciones para obtener resultados lo más representativos posible. Cada ejecución se termina cuando convergen los memes, lo que por simplicidad se considera que ha sucedido cuando todos los memes son iguales en al menos las dos primeras posiciones decimales.

En primer lugar se analiza la propagación de los memes como una función del potencial de mejora relativa de los memes al principio de la ejecución. Para este propósito, se toma $D = [0,1]$ y se considera un escenario en el que los genotipos y los memes se inicializan aleatoriamente en este rango, y otro escenario en el que los genotipos toman valores iniciales en el rango $[0,0{,}5]$, mientras que los memes se muestrean aleatoriamente en el rango $[0,1]$. La Figura 3.2 muestra la distribución de los memes en cada instante de tiempo (cuanto más cálido –rojo– es el color más alto es el valor del meme). Centrando la atención en la fila superior (topología panmíctica), se puede apreciar claramente el diferente comportamiento dependiendo de la inicialización del genotipo. Cuando tanto los genes como los memes se inicializan ambos en el rango $[0,1]$, el algoritmo raramente converge a un meme de alto potencial. En realidad, tales memes proliferan temporalmente en las etapas iniciales del algoritmo, pero después son conducidos a la extinción por los memes que se han unido a genotipos de alta calidad. La situación es bastante diferente cuando los genotipos se inicializan dentro del rango $[0,0{,}5]$: en este caso el algoritmo converge gradualmente a la parte más alta de la distribución de memes, con los memes de bajo potencial desapareciendo rápidamente de la población. Una perspectiva más detallada sobre esto se proporciona a través de la Figura 3.3 en la que se muestran Distribuciones Cualificadas de las Ejecuciones (QRTDs)[2] [177]. Estas indican la probabilidad de que se alcance un determinado objetivo (en este caso, la convergencia a un meme en el percentil deseado) en función del número de iteraciones. Se puede observar como las probabilidades están por debajo del 10 % para los memes a partir del percentil 95 % en el primer escenario, mientras que esta probabilidad llega incluso al 100 % en el segundo escenario. Ello se debe a que en la última no puede pasar una coincidencia espuria entre un muy buen genotipo y un mal meme, ya que dichas soluciones muy buenas no existen inicialmente. Además, los memes de alto potencial que se fusionan inicialmente con genotipos malos pueden fácilmente mejorar la calidad de estos últimos en las etapas iniciales y por lo tanto, incrementar sus posibilidades de supervivencia.

Centrando ahora la atención en el efecto de la estructura espacial, la fila

[2] *qualified runtime distribution*

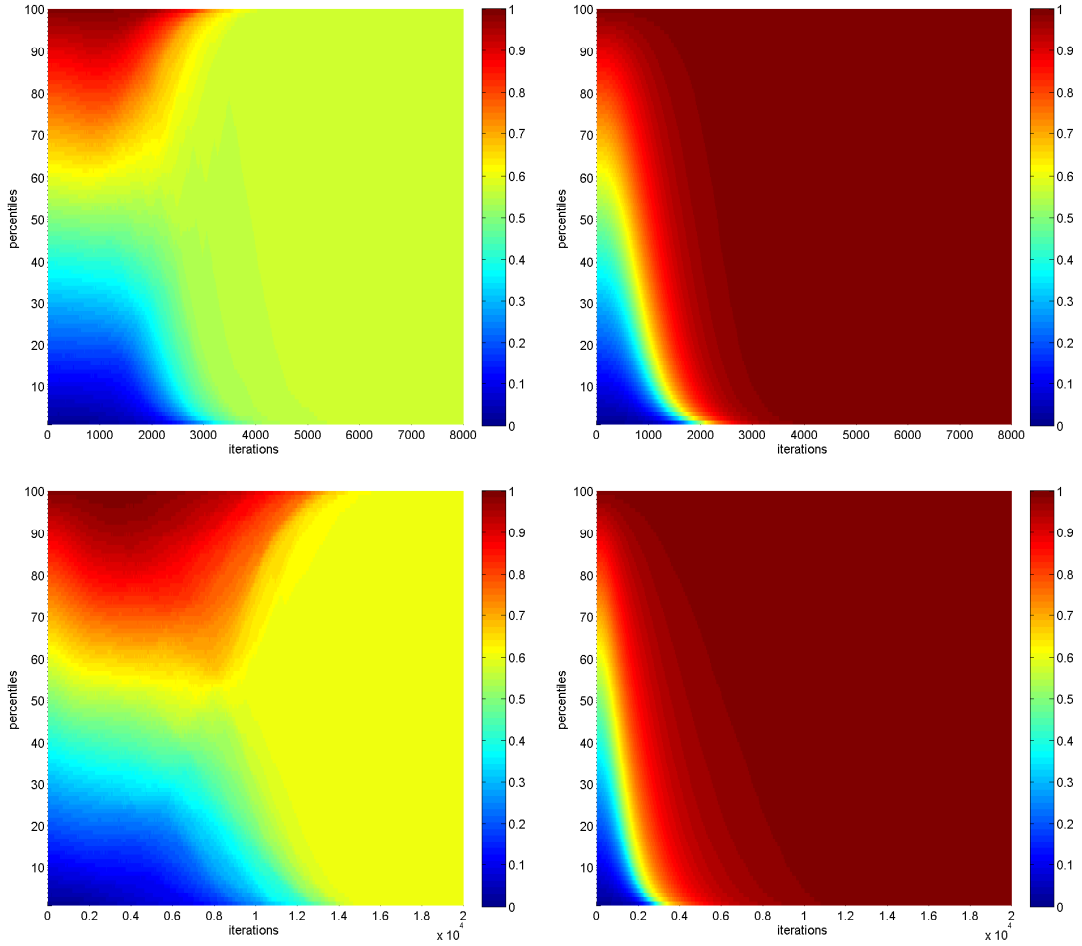


Figura 3.2: Mapas de memes para simulaciones con $\mu = 625$. La fila superior corresponde a conectividad panmíctica y la inferior a un vecindario de von Neumann. Del mismo modo, la columna izquierda corresponde a genotipos inicializados en el rango $[0, 1]$ y la de la derecha a inicialización en $[0, 0{,}5]$ (los memes se inicializan en el rango $[0, 1]$ en ambos casos). Los tonos más cálidos (rojos) corresponden a los valores de los memes más altos. La evolución del algoritmo se representa en cada subfigura de izquierda a derecha, donde cada franja vertical representa la distribución de memes para un cierto instante de tiempo t. Considérese la diferencia de escala en el eje X.

inferior de la Figura 3.2 muestra la distribución de los memes para el caso de vecindad de von Neumann. Se puede observar también un patrón similar al del caso panmíctico con respecto a la inicialización del genotipo. Sin embargo, una inspección más detallada revela varias diferencias. En primer lugar, se observa como la convergencia es más lenta en este caso (e.g., la escala en el eje X es mayor). Este es un efecto bien conocido del uso de estructuras espaciales y se aprovecha habitualmente en el contexto de los algoritmos evolutivos para promocionar la

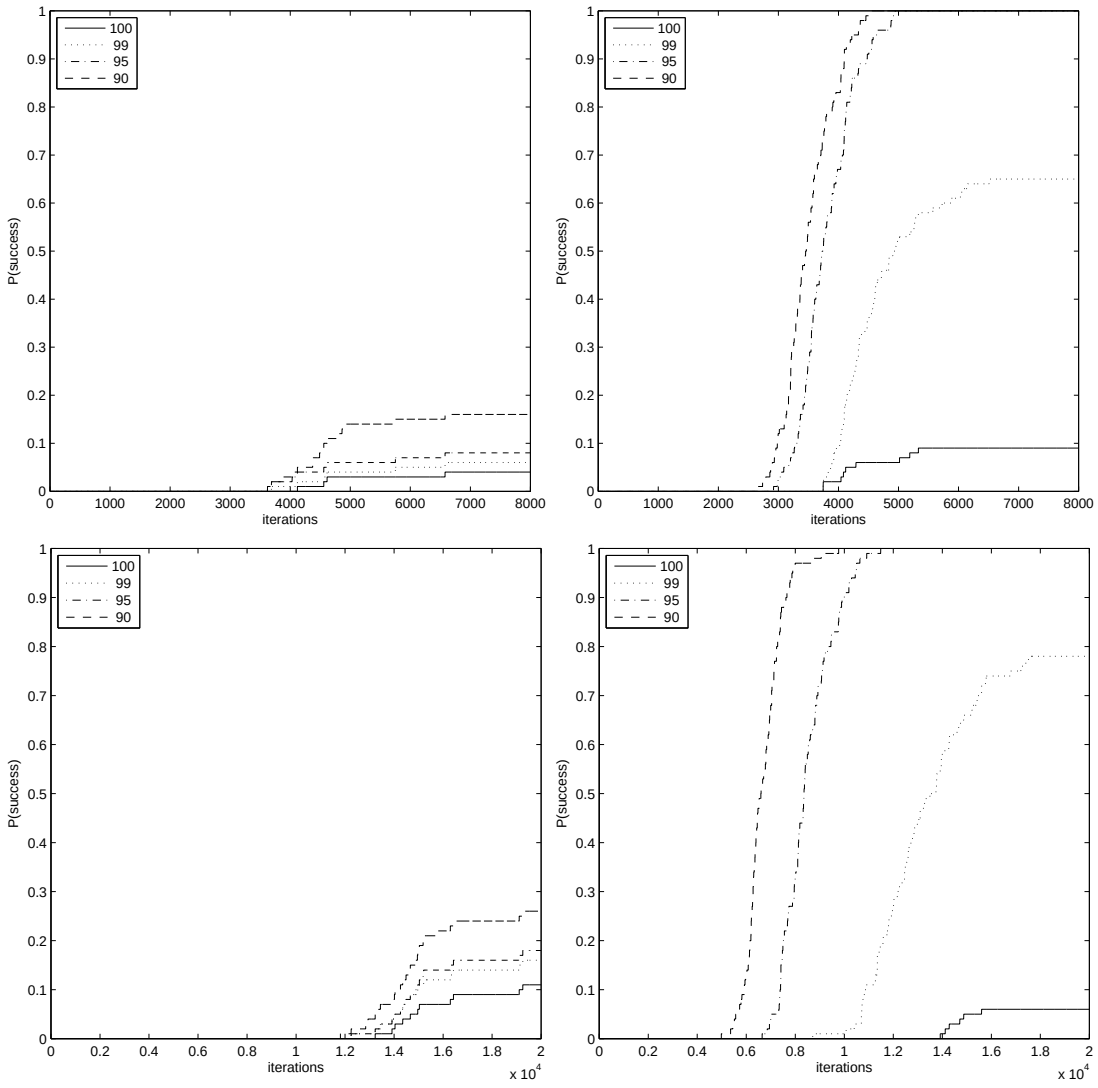


Figura 3.3: QRTDs para simulaciones con $\mu = 625$. La fila superior corresponde a conectividad panmíctica y la inferior a un vecindario de von Neumann. Del mismo modo, la columna izquierda corresponde a genotipos inicializados en el rango $[0, 1]$ y la de la derecha a inicialización en $[0, 0{,}5]$ (los memes se inicializan en el rango $[0, 1]$ en ambos casos). Las curvas indican la probabilidad de que la población converja a un meme en el percentil inicial i-ésimo de la población en función del número de iteraciones. Considérese la diferencia de escala en el eje X.

diversidad y por lo tanto decrementar la probabilidad de que se atasquen en un óptimo local [97, 348]. En el caso de los MMAs, esto tiene una ventaja adicional: una convergencia más lenta incrementa la vida útil de los memes individuales y por lo tanto, se les ofrece más oportunidades para mejorar los individuos que los contienen, evidentemente si tienen el potencial para hacerlo. Por lo tanto, el algoritmo es más robusto y puede hacer frente mejor a los memes inertes (aquellos

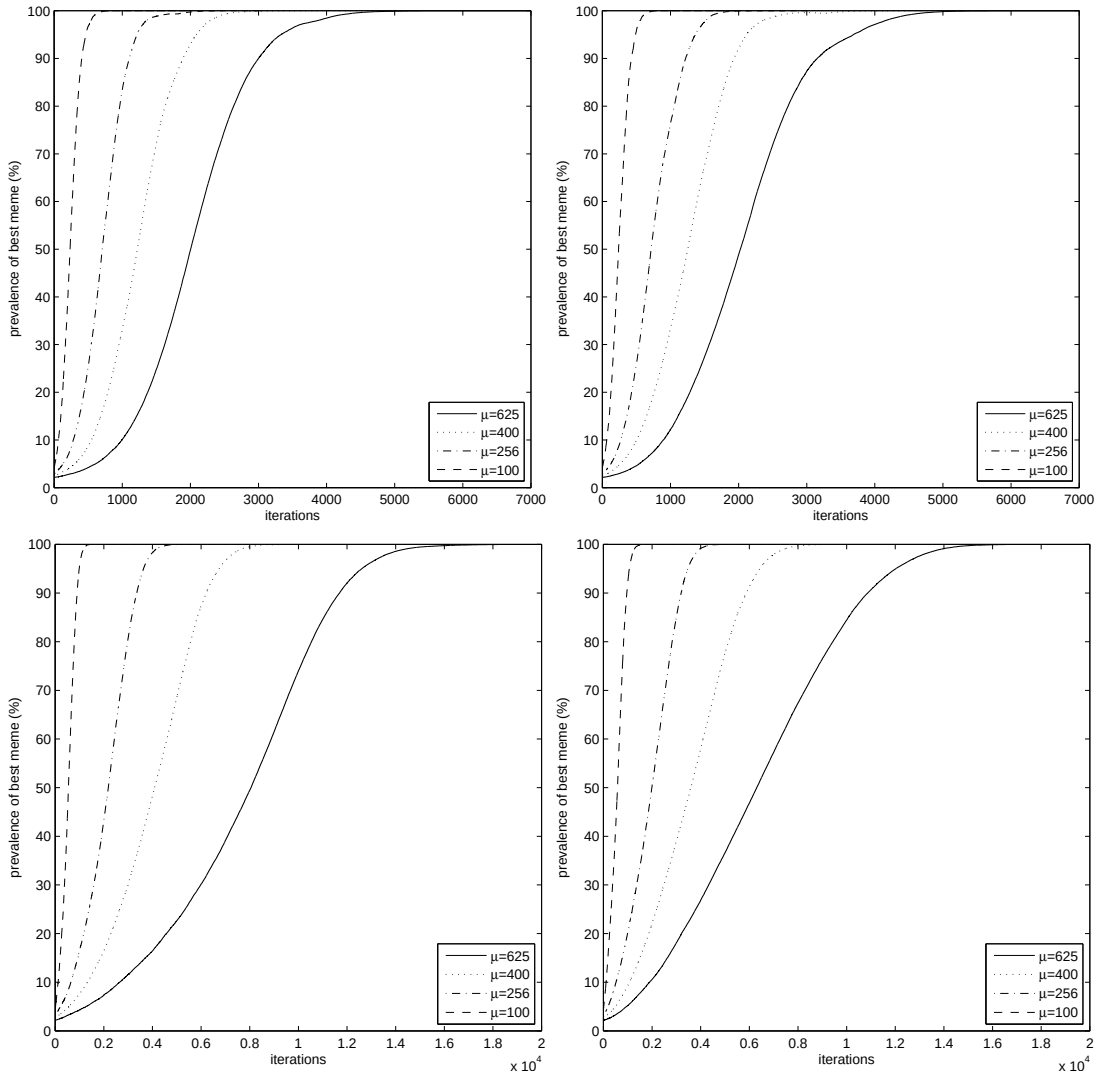


Figura 3.4: Curvas de crecimiento para diferentes tamaños de población. La fila superior corresponde a conectividad panmíctica y la inferior a un vecindario de von Neumann. Del mismo modo, la columna izquierda corresponde a genotipos inicializados en el rango $[0, 1]$ y la derecha a una inicialización en $[0, 0{,}5]$ (los memes se inicializan en el rango $[0, 1]$ en ambos casos). Considérese la diferencia de escala en el eje X.

que son de baja calidad pero se unen a buenos genotipos). Esto se puede ver en la fila inferior del mapa de memes (ver Figura 3.2), donde hay un gran predominio de áreas de colores cálidos, y más claramente en las QRTDs (fila inferior de la Figura 3.3), donde por ejemplo el percentil 95 % se alcanza con una probabilidad de cerca del 20 % en el caso de inicialización en el rango $[0, 1]$ (compárese con una probabilidad inferior al 10 % en el caso panmíctico) y el percentil 99 % se alcanza con una probabilidad cercana al 80 % para la inicialización en el rango

Tabla 3.1: Ajustes de las curvas de crecimiento a la función logística. Se muestra para cada algoritmo y configuración la cota superior usada para la inicialización de memes (G), el parámetro de escala (α) y el error cuadrático medio (mse).

		population size			
		$\mu = 100$		$\mu = 256$	
topology	G	α	mse	α	mse
panmictic	0.5	81.628878	0.000042	210.791430	0.000069
	1.0	77.708284	0.000008	187.904342	0.000030
von Neumann	0.5	186.815036	0.000366	578.914610	0.000554
	1.0	168.567695	0.000322	585.804249	0.000532
		$\mu = 400$		$\mu = 625$	
topology	G	α	mse	α	mse
panmictic	0.5	324.649846	0.000064	521.933265	0.000017
	1.0	297.902782	0.000027	462.866673	0.000010
von Neumann	0.5	1046.429102	0.000361	1974.995403	0.000318
	1.0	1057.519394	0.000537	1945.625376	0.000705

$[0,0{,}5]$ (compárese con alrededor del 65 % de probabilidad del caso panmíctico). Un test signrank [289, 365] indica que la diferencia alcanzada en el percentil final es estadísticamente significativa en ambos casos ($\alpha = 0{,}05$).

Finalmente se va a considerar el tiempo de absorción, concretamente el tiempo necesario para que un meme (no necesariamente el mejor como se mostró previamente) domine completamente la población. La Figura 3.4 muestra las curvas de crecimiento, representando el porcentaje del conjunto de memes ocupado por el meme más repetido (recalcar que el meme más repetido no es necesariamente el mismo a lo largo de una ejecución; simplemente se cuenta el número de copias del más repetido en cada iteración). Estas curvas muestran la típica forma del conocido modelo logístico $f(t) = 1/(1 + Ke^{-t/\alpha})$. Este modelo fue propuesto inicialmente en la literatura por Sarma y De Jong [319] en el contexto de los EAs con estructura espacial. A pesar de que esta no tiene por qué ser la única alternativa –e.g., ver [146]– sí sirve como un buen punto de partida para cuantificar el crecimiento de los memes dominantes. Cualitativamente, se observa como era de esperar el conocido patrón de una convergencia más lenta conforme se incrementa el tamaño de la población y para la topología de von Neumann [145] como contrapartida a las poblaciones panmícticas. Desde un punto de vista cuantitativo, se ha ajustado el crecimiento de los datos a una curva logística para identificar el factor de escala α. Los datos resultantes se muestran en la Tabla 3.1. Como se puede ver, el ajuste es bastante bueno, teniendo unos errores cuadráticos medios muy bajos. Los parámetros de escala son bastante similares para las variantes con la misma topología, alrededor de entre 2 y 5 veces mayores para la topología de von Neumann que para el caso panmíctico, lo que encaja con los tiempos de absorción, como se puede ver en la Figura 3.4. Con respecto al tamaño de la población, el incremento del parámetro de escala admite una interpolación lineal

$\alpha = a + b\mu$ obteniéndose valores de $b = 0{,}84$ y $b = 0{,}74$ para el caso panmíctico y de $b = 3{,}43$ y $b = 3{,}40$ para la rejilla con topología con conectividad de von Neumann.

3.2.4. Resultados Experimentales

Una vez analizado el modelo idealizado de los MMAs, ahora toca centrarse en una versión algorítmica operativa de los mismos a efectos de validar los hallazgos previos. El MMA considerado se describirá a continuación y posteriormente se presentarán y analizarán los resultados experimentales.

3.2.4.1. Resultados

Los experimentos se han considerado sobre el MMA descrito en la Sección 3.1, usando un tamaño de población de $\mu = 256$ individuos, selección por torneo binario, probabilidad de recombinación $p_X = 1{,}0$ y probabilidad de mutación $p_M = 1/\ell$ donde ℓ es la longitud de los genotipos. La recombinación del genotipo se realiza usando un operador de cruce de un punto. Con respecto a los memes, tienen una probabilidad p_X de que los descendientes hereden el meme transportado por el mejor de sus padres (en otro caso heredan el meme de su primer padre). Los memes se definen como reglas de longitud $r = 3$ y se aplican con el parámetro $w = 1$ (se realiza una reescritura y se mantiene si se mejora el genotipo). Se considera un número máximo de 25000 evaluaciones para las funciones TRAP y H-IFF de 128 bits y 50000 evaluaciones para el problema MMDP de 144 bits (ver detalles de los problemas en el Apéndice A). En todos los casos el coste de la aplicación del meme se contabiliza como una evaluación parcial (como la fracción de los subproblemas que es necesario reevaluar después de modificar el genotipo), por lo tanto, se realiza una medida del sobrecoste que conlleva la aplicación de los memes. Al igual que en la Sección 3.2.3, se han considerado dos variantes algorítmicas: un MMA con población panmíctica y con estructura espacial de la población (rejilla cuadrada toroidal con topología de von Neumann). Todos los experimentos con todos los algoritmos y todos los problemas de test se han repetido 100 veces.

En primer lugar se observa la evolución del mejor fitness en la Figura 3.5. El MMA panmíctico (PANM) tiene una convergencia inicial más rápida pero pronto es sobrepasado por el MMA de von Neumann (VN) que consigue mantener de forma constante la mejora durante más tiempo, alcanzando finalmente soluciones notablemente mejores –ver la Tabla 3.2 para una descripción más detallada de los datos numéricos. Esta mejora es consistente en todos los problemas del test y se puede mostrar que es estadísticamente significativa en el nivel de $\alpha = 0{,}05$ usando el test de ranksum de Wilcoxon [289, 365]. Una perspectiva complementaria sobre esta convergencia mejorada del VN se muestra en la Figura 3.6. Inspeccionando las QRTDs se puede observar cómo hay una mejora remarcable en la probabilidad de éxito en el caso de VN con respecto a PANM en los tres problemas considerados.

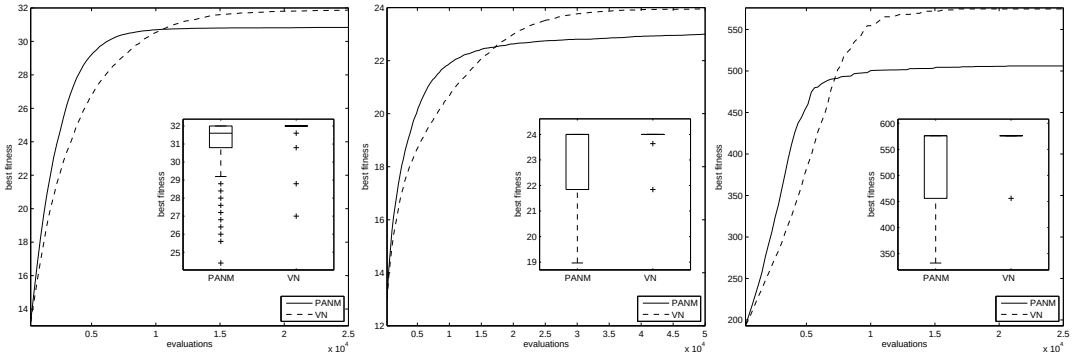


Figura 3.5: Curvas de mejor fitness (promediado sobre 100 ejecuciones) de los MMAs con conectividades panmícticas y von Neumann. El gráfico interno representa la distribución de los valores de fitness alcanzados. De izquierda a derecha: TRAP, MMDP y H-IFF. Considérese la diferencia de escala en el eje X.

Tabla 3.2: Resultados (promediados sobre 100 ejecuciones) de los MMAs con conectividades panmíctica y von Neumann. Se muestra el número de veces que se alcanza el óptimo (n_{opt}), la mediana ($\tilde{x}$), la media ($\bar{x}$) y el error estándar de la media ($\sigma_{\bar{x}}$).

	TRAP			MMDP			H-IFF		
	n_{opt}	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$	n_{opt}	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$	n_{opt}	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$
panmictic	49	31.6	30.8 ± 0.2	60	24.0	23.0 ± 0.1	54	576.0	506.1 ± 8.1
von Neumann	91	32.0	31.9 ± 0.1	97	24.0	24.0 ± 0.0	99	576.0	574.8 ± 1.2

No solo VN es capaz de encontrar las soluciones óptimas más a menudo (la tasa de éxito para VN es del 91 %-99 % frente al 49 %-60 % para PANM) sino que también es capaz de aproximarse a las soluciones de alta calidad de forma más habitual. Esto es congruente con los hallazgos del modelo idealizado y puede ser debido a la mayor diversidad de los memes, que son capaces de mantener una fructífera búsqueda lamarckiana durante más tiempo. La evolución de la diversidad (medida en términos de la entropía de los memes de la población) se representa en la Figura 3.7 (b). La diversidad memética decrece más suavemente en el caso de VN, lo que permite que potencialmente buenos memes tengan más oportunidades de progresar (básicamente a partir de la mejora de la calidad de los resultados, como se mencionó con anterioridad). Esto afecta a su vez de forma indirecta a los individuos a nivel genético, haciendo que también haya una mayor diversidad genotípica, como se puede comprobar en la Figura 3.7 (a) cuyas curvas tienen un aspecto muy similar a las meméticas. De hecho, como se muestra en la Figura 3.8, la tasa de éxito de la aplicación de los memes es más alta en el caso de MMAs no panmícticos, lo que ayuda a explicar el mayor rendimiento del fitness de estos últimos.

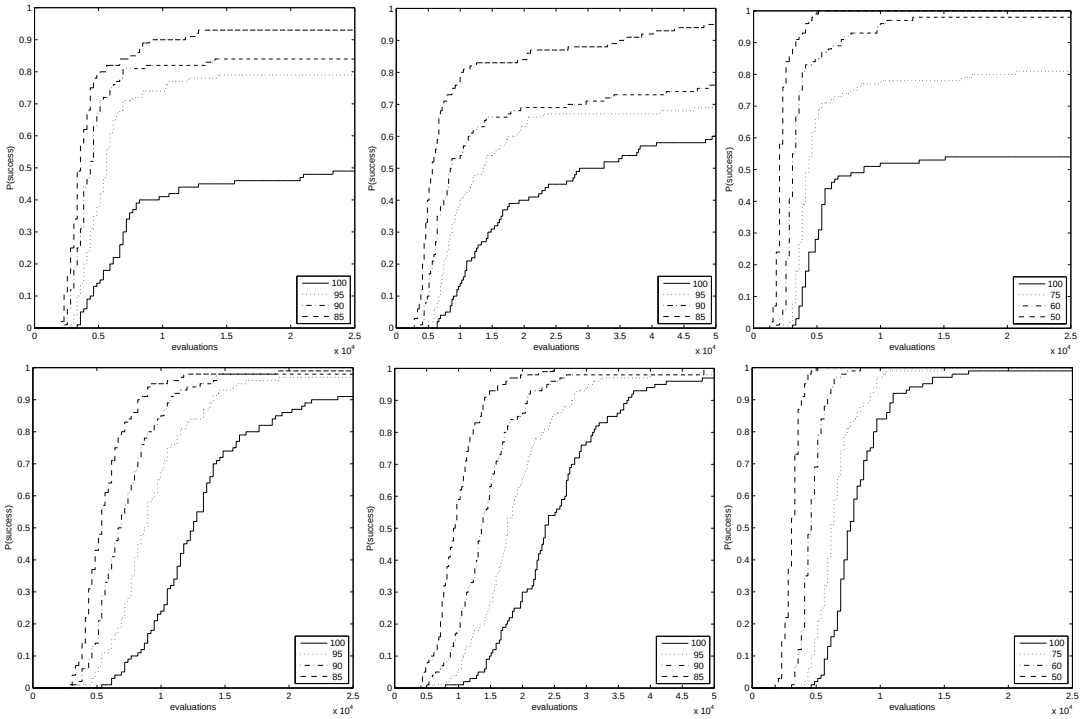
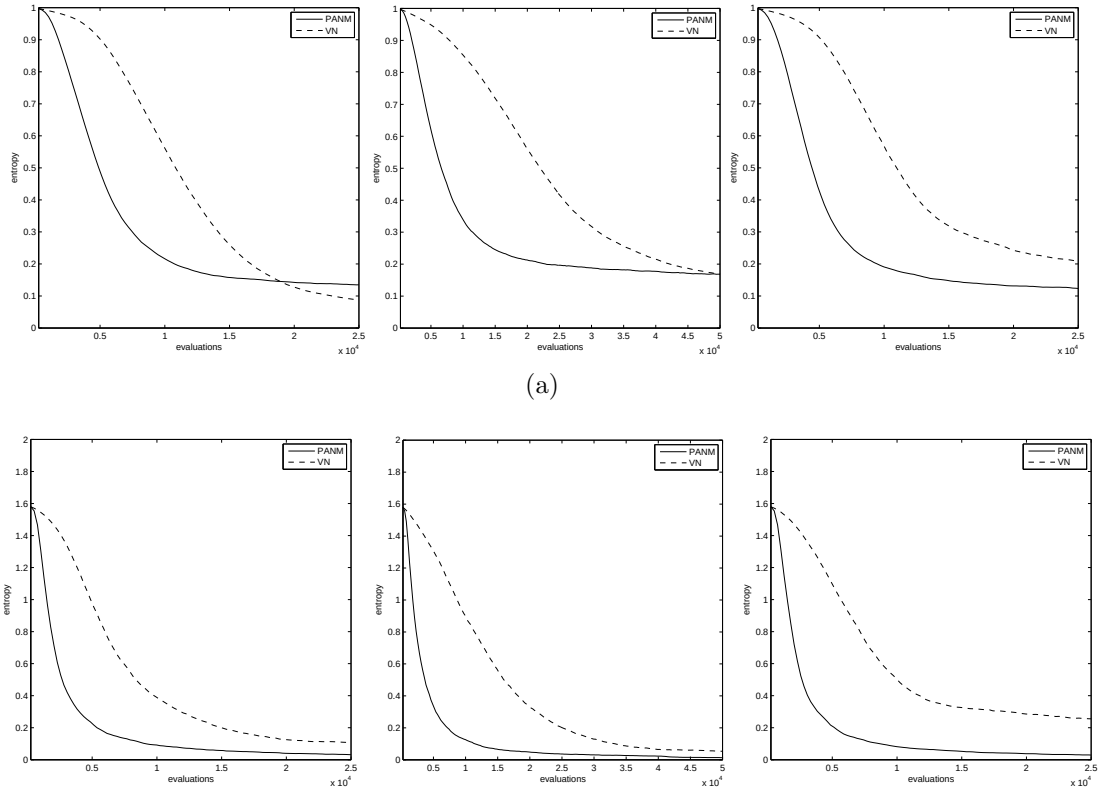


Figura 3.6: QRTDs para diferentes variantes del MMA. La fila superior corresponde a conectividad panmíctica y la inferior a un vencindario de von Neumann. Por otro lado, la columna de más a la izquierda corresponde a la función TRAP, la del medio a MMDP y la de más a la derecha a H-IFF. Las curvas indican la probabilidad de que la población converja a una solución cuya calidad esté en el percentil i-ésimo del óptimo en función del número de iteraciones. Estos porcentajes de aproximación son diferentes en el caso de H-IFF debido a las particularidades de la distribución de valores en esta función (huecos mayores entre la solución óptima y las subóptimas). Considérese la diferencia de escala en el eje X.

3.2.5. Discusión

El análisis de la propagación de memes en los MMAs usando un modelo idealizado de genotipos y memes ha mostrado que la intensidad de la selección juega un papel muy importante en el hecho de que los memes de alto potencial tengan la oportunidad de proliferar en la población. En el modelo panmíctico, los memes buenos dominarán la población final cuando las soluciones de partida tengan un margen de mejora sustancial con respecto a la media. Cuando este margen es más pequeño, los memes promedio pueden por azar mantenerse hasta las etapas finales de la evolución y hacer que otros comparativamente mejores lleguen a extinguirse. En presencia de estructuras espaciales que inducen tiempos de absorción más largos (en este caso, una rejilla cuadrada toroidal con topología de von Neumann), este efecto de comensalismo se mitiga de alguna manera permitiendo



(a)

(b)

Figura 3.7: Evolución de la diversidad genética (a) y memética (b) para diversos problemas con $\mu = 256$ y conectividades de tipo panmíctico y von Neumann. Las columnas izquierdas corresponden a la función TRAP con 128 bits, las centrales al problema MMDP con 144 bits y las de la derecha al problema H-IFF con 128 bits. Considérese la diferencia en la escala del eje X.

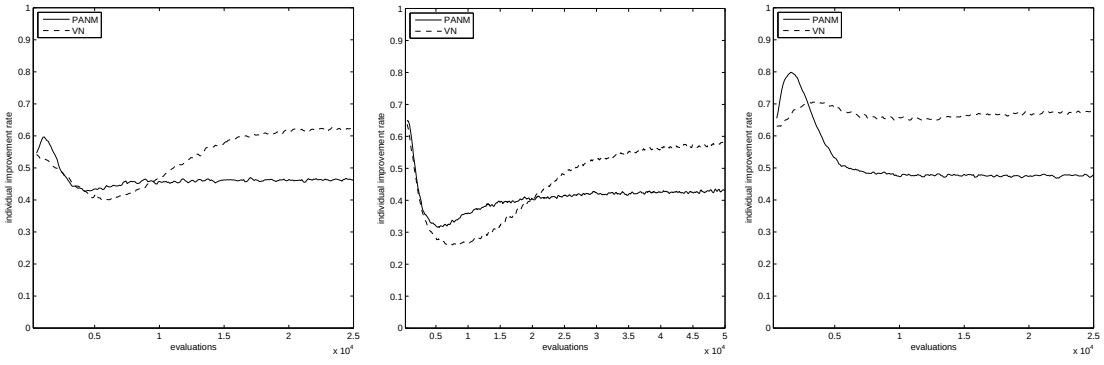


Figura 3.8: Evolución de la tasa de éxito de los memes (fracción de las veces que la búsqueda local produce una mejora en un individuo concreto). De izquierda a derecha: TRAP, MMDP y H-IFF. Considérese la diferencia de escala en el eje X.

que los memes de mayor calidad se mantengan e incrementen sus tasas de supervivencia y por lo tanto sus posibilidades de permanecer en la población hasta las etapas finales de la evolución a la vez que tienen la oportunidad de expandirse mejorando asimismo los individuos que los contienen. Por otro lado, el MMA con población con estructura espacial se ha mostrado que mantiene una diversidad memética más elevada y que puede proporcionar mejores resultados con respecto a sus equivalentes panmícticos.

3.3. Auto-Adaptación en Algoritmos Multimeméticos

En esta sección se analiza la auto-adaptación en el contexto de los MMAs con estructura espacial, en los que se dota a la población de una cierta topología para restringir las interacciones (desde el punto de vista de la selección y de los operadores de variación) con el objetivo de estudiar la influencia de dichas estructuras en el proceso de auto-adaptación.

3.3.1. Esquema auto-adaptativo

La principal ventaja de tener memes enlazados a los individuos es que dota al algoritmo de la habilidad de descubrir las definiciones de vecindad más apropiadas para una solución determinada, así como una efectividad mayor en la exploración de dichos vecinos. Estos vecindarios pueden evolucionar a través de las soluciones, proporcionando una auto-adaptación que permite estimular el proceso de búsqueda, de tal forma que estos mecanismos de mejora local sean capaces de adaptarse dinámicamente. Esta auto-adaptación no está limitada a la definición actual del vecindario para un *radio* fijo determinado (i.e., distancia de Hamming), sino que puede involucrar al radio en sí mismo. Para hacer esto, la longitud r de los memes se define dentro de un intervalo determinado $\{l_{\text{mín}}, \cdots, l_{\text{máx}}\}$. Inicialmente, cada meme tiene una longitud aleatoria en este rango. A continuación, en cada ciclo evolutivo y antes de que un cierto meme vaya a mutar y por lo tanto a aplicarse sobre un genotipo, su longitud se puede incrementar o disminuir en una unidad como mucho [334]. Esto se realiza con una probabilidad determinada p_r (en el caso del incremento de la longitud del meme se añade un nuevo símbolo de forma aleatoria en la posición del extremo derecho; si la longitud se decrementa se elimina del meme el símbolo de más a la derecha). Haciendo esto, la longitud de las reglas de reescritura se puede ajustar dinámicamente mediante la evolución, lo que proporciona un control auto-adaptativo de su complejidad: los memes grandes son una herramienta poderosa para realizar grandes saltos en el espacio de búsqueda, pero por otro lado, al ser más específicos pueden tener una menor tasa de aplicabilidad. Esto permite al algoritmo ser capaz de descubrir la complejidad apropiada de los memes en cada momento.

En esta sección el punto de interés se centra en el uso de memes de complejidad auto-adaptativa en combinación con las poblaciones con estructura espacial, analizando comparativamente su efectividad en este contexto.

3.3.2. Análisis Experimental

Para analizar el impacto de la auto-adaptación de los memes sobre los MMAs descritos en la sección previa se han considerado dos problemas de optimización pseudobooleana como son TRAP y H-IFF (ver Apéndice A), para a continuación analizar los resultados obtenidos durante los experimentos.

Se considera un MMA como el descrito en la Sección 3.1, con un tamaño de población de $\mu = 100$ individuos. Este MMA sigue un plan reproductivo generacional con torneo binario para la selección de los padres, cruce de un punto ($p_X = 1{,}0$), mutación bit-flip ($p_M = 1/\ell$, donde $\ell = 128$ es el número de bits), búsqueda local (dirigida utilizando memes enlazados a los individuos) y reemplazo del peor padre (una estrategia elitista de herencia, siguiendo el modelo presentado en [269]). Los descendientes heredan el meme del mejor padre, el cual está sujeto posteriormente a mutación con probabilidad p_M. Cada ejecución finaliza cuando se alcanzan 25000 evaluaciones, realizándose 20 ejecuciones por cada problema y algoritmo. Se considera que la longitud de los memes está limitada en el rango $l_{\text{mín}} = 3$ y $l_{\text{máx}} = 9$, usándose una probabilidad $p_r = 1/l_{\text{máx}}$ para la auto-adaptación en longitud de las reglas. Con el propósito exclusivo de establecer comparaciones, se han considerado memes de longitud fija de tamaño ($r \in \{3, 6, 9\}$) y tres variantes algorítmicas: un MMA con población panmíctica y dos más con estructura espacial de la población (uno de ellos con rejilla cuadrada toroidal con topología de von Neumann y el otro con topología de Moore). Para los MMAs con estructura espacial se utiliza una rejilla de tamaño 10×10 y un radio para el vecindario $\rho = 1$.

Los resultados numéricos de los diferentes MMAs se muestran en la Tabla 3.3. Cualitativamente, los MMAs panmícticos (con independencia de las longitudes de los memes) parecen funcionar comparativamente peor que sus correspondientes versiones de Moore y von Neumann, lo que avala el impacto positivo que tiene la convergencia más lenta inducida por las estructuras espaciales sobre los resultados finales. Esto además es congruente con la ligera superioridad del MMA con topología de von Neumann sobre el MMA con topología de Moore, el cual tiene una tasa de convergencia más rápida, tanto a nivel genotípico como memético – ver la Figura 3.9.

Centrándose ahora en el efecto de las longitudes de los memes, si en primer lugar se observan los resultados usando memes de longitud fija parece que el valor intermedio $r = 6$ ofrece el mejor compromiso entre la riqueza memética y la especificidad de los memes para los valores considerados. Esto ofrece una primera estimación de la forma en la que las longitudes de los memes auto-evolucionan. Ciertamente, si se observa la Figura 3.10, se puede comprobar que las longitudes medias de los memes oscilan alrededor del valor 6, lo que parece indicar que

Tabla 3.3: Resultados (20 ejecuciones) de los diferentes MMAs sobre los dos problemas considerados. También se muestra el número de veces que se encuentra el óptimo (n_{opt}), la mediana ($\tilde{x}$), la media ($\bar{x}$) y el error estándar de la media (σ_x).

		TRAP			H-IFF		
topology	r	n_{opt}	$\tilde{x}$	$\bar{x} \pm \sigma_x$	n_{opt}	$\tilde{x}$	$\bar{x} \pm \sigma_x$
panmictic	3	5	30.4	29.5 ± 0.5	1	390.0	404.1 ± 14.9
	6	9	30.4	29.6 ± 0.5	4	382.0	420.3 ± 20.4
	9	3	28.4	28.7 ± 0.5	2	362.0	382.0 ± 16.6
	$3-9$	8	29.0	29.2 ± 0.6	8	456.0	475.3 ± 20.8
Moore	3	8	31.2	30.4 ± 0.4	5	444.0	460.0 ± 17.0
	6	10	31.2	30.0 ± 0.5	8	456.0	476.4 ± 19.7
	9	7	29.8	29.8 ± 0.4	6	456.0	449.0 ± 21.4
	$3-9$	11	32.0	30.8 ± 0.4	7	460.0	471.8 ± 19.5
von Neumann	3	6	30.8	30.1 ± 0.5	9	464.0	501.0 ± 16.9
	6	13	32.0	30.9 ± 0.4	12	576.0	518.4 ± 17.0
	9	10	31.8	30.4 ± 0.4	9	464.0	491.7 ± 18.6
	$3-9$	15	32.0	31.2 ± 0.3	12	576.0	515.6 ± 18.1

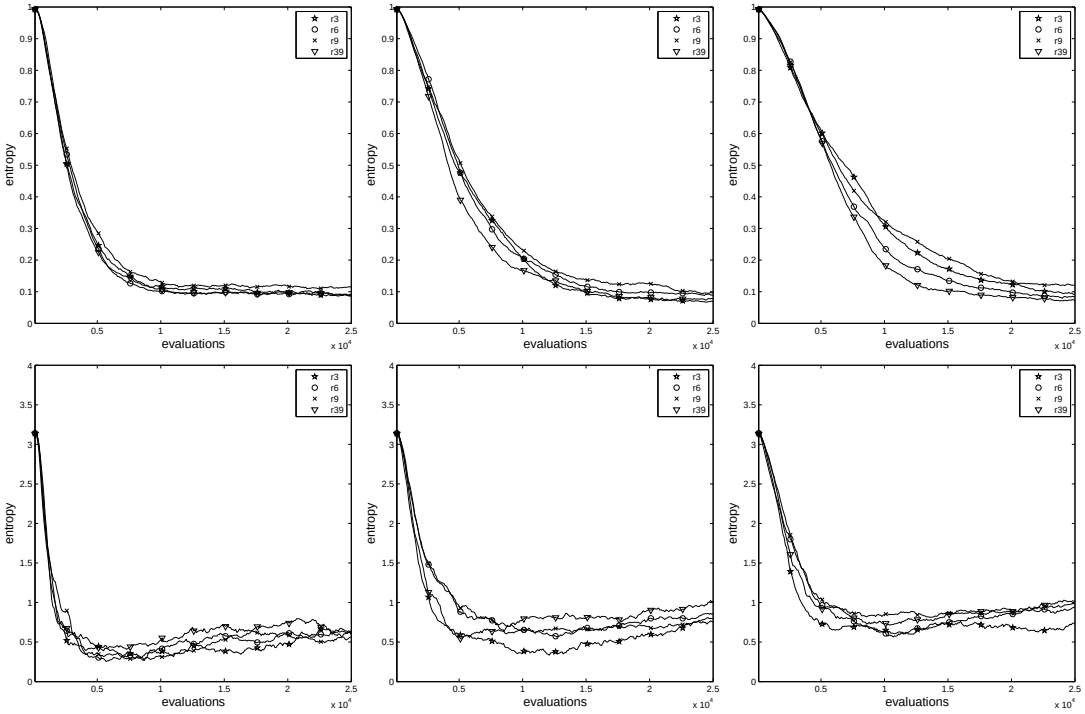


Figura 3.9: Evolución de la diversidad de diferentes MMAs sobre la función TRAP. La fila superior corresponde a diversidad genética y la inferior a diversidad memética. Las topologías utilizadas de izquierda a derecha son: panmíctica, Moore y von Neumann.

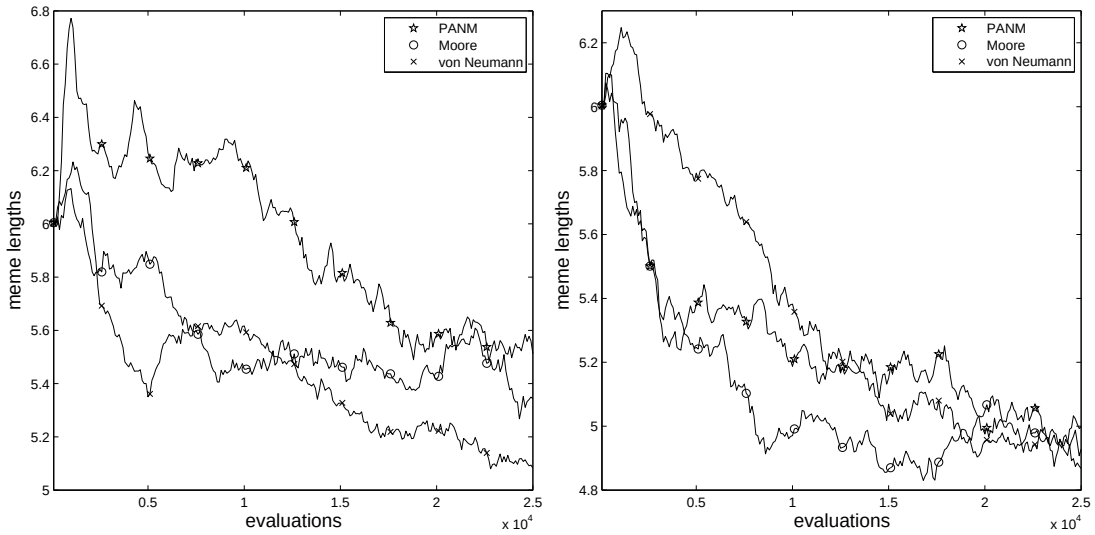


Figura 3.10: Evolución de las longitudes de los memes en MMAs auto-adaptados con diferentes topologías. (Izquierda) TRAP. (Derecha) H-IFF.

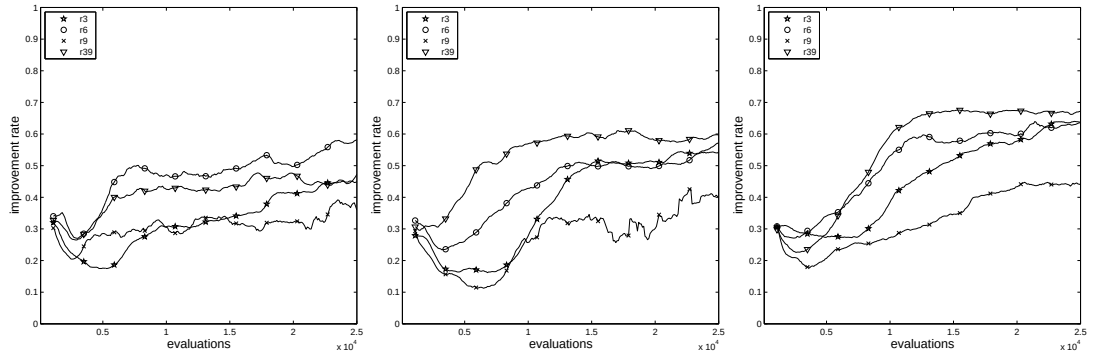


Figura 3.11: Tasa de éxito de los memes (porcentaje de aplicaciones de memes que producen una mejora en el individuo correspondiente) de los diferentes MMAs sobre la función TRAP. Topologías utilizadas de izquierda a derecha: panmíctica, Moore, y von Neumann.

el MMA totalmente auto-adaptativo considera que esta región tiene un elevado interés memético. Esto se puede justificar más a fondo con los resultados del fitness y por el número de veces que cada algoritmo encuentra el óptimo: el MMA_{3-9} funciona análogamente o incluso mejor que el MMA_6 (la diferencia está más marcada a favor del MMA_{3-9} en el caso de población panmíctica y H-IFF). Se ha pasado el test estadístico de Quade a todos los problemas, todas las topologías y todas las longitudes de reglas, tanto en función de la aproximación al valor óptimo obteniendo un p-valor $= 1{,}97 \cdot 10^{-3}$, como con respecto al número de veces que cada algoritmo encuentra el óptimo, cuyo p-valor es $6{,}40 \cdot 10^{-4}$, muy próximos en ambos casos a 0, concluyendo que hay diferencias significativas

entre las distintas variantes. Por otro lado, la tendencia general decreciente de las longitudes de los memes en este MMA_{3-9} es un fenómeno interesante. Una posible conjetura es que durante el progreso de la evolución el algoritmo empieza a localizar óptimos o soluciones próximas al óptimo y el papel de los memes podría estar cambiando de ser un artefacto eminentemente de búsqueda (intentando encontrar direcciones de búsqueda hacia el valor óptimo) a una función con un mecanismo de corrección de errores (corrigiendo las perturbaciones introducidas por la mutación sobre las soluciones (cuasi)-óptimas), es decir, su papel da un giro pasando de ser exploratorio a explotador.

Esta interpretación es congruente con la tasa de éxito de los memes (porcentaje de aplicaciones de los memes que producen un resultado mejorado), como se muestra en la Figura 3.11. Considérese también que esto sigue una tendencia ascendente (y que los valores para el MMA_{3-9} son normalmente superiores a los restantes MMAs, en particular para las poblaciones no panmícticas), lo que podría estar indicando este papel activo como resultado de la corrección del error (los valores de fitness son bastante más estables en las etapas de evolución posteriores, y por lo tanto una tasa de éxito elevada podría interpretarse como que soluciones de baja calidad están siendo reparadas para volver a intentar encontrar soluciones mejores o incluso el valor óptimo). La evolución de la diversidad también es representativa en este sentido –como se puede ver en la Figura 3.9– ya que la diversidad genética parece decrementarse más rápido en el MMA totalmente auto-adaptativo (más claramente en el caso de topología von Neumann), mientras la diversidad memética permanece comparativamente más elevada.

3.3.3. Discusión

Como es bien conocido, la parametrización es una de las cuestiones fundamentales dentro del paradigma de los algoritmos meméticos [343], incluso más si se considera los MMAs, los cuales necesitan parámetros adicionales para controlar la representación de los memes. Por esta razón, el estudio de mecanismos de auto-adaptación que alivien este problema de parametrización tiene un interés primordial. Los resultados obtenidos con MMAs con estructura espacial y con auto-adaptación a través de memes de longitud variable sobre los problemas considerados y las tres topologías analizadas (panmíctica, Moore y von Neumann) indican que la auto-adaptación de las longitudes de los memes no es perjudicial e incluso en algunas ocasiones es beneficiosa, principalmente en el caso de poblaciones panmícticas. Este último efecto puede atribuirse a que los MMAs no panmícticos tienen una parametrización más robusta cuando se encuentran con subóptimos. En cualquier caso, globalmente la auto-adaptación a través de las longitudes de los memes parece una estrategia adecuada para los problemas que se han considerado, ya que no penaliza el rendimiento y permite ahorrar en tiempo de configuración.

3.4. Algoritmos de Estimación de Distribuciones Multimeméticos

Los MMAs realizan habitualmente la evolución memética usando los procedimientos de recombinación y mutación estándar de los EAs para manipular directamente la información a nivel memético. En esta sección se va a considerar el uso de EDAs [186, 218, 298, 299] como motor de búsqueda subyacente para la optimización multimemética. Mientras el uso de procedimientos de búsqueda local en combinación con EDAs es una aproximación ampliamente estudiada para inyectar conocimiento dependiente del problema y mejorar la eficiencia del proceso de optimización –ver por ejemplo en [296, 303, 313, 371]– el uso de EDAs para optimización memética auto-adaptativa se ha estudiado con menos profundidad. La contribución de este capítulo es avanzar algunos pasos en esta dirección, estudiando diferentes aproximaciones para la aplicación de EDAs a la optimización multimemética y proporcionando una evaluación empírica extensa del rendimiento de esta aproximación.

3.4.1. Aproximación Probabilística

La idea subyacente en el modelo considerado para el MMA es disponer de información genética y memética enlazada dentro de cada individuo concreto, es decir, cada individuo transporta un genotipo y un meme. Una vez se ha generado un individuo, se evalúa su genotipo y a continuación se aplica el meme para mejorarlo. Considerando que estos individuos se generan a través de operadores evolutivos –tales como la recombinación y la mutación– en MMAs genéticos estándar, los EDAs multimeméticos aproximan este funcionamiento a través de un muestreo probabilístico de una cierta distribución que evoluciona durante la ejecución.

Los EDAs intentan aprender la distribución de probabilidad $p(\vec{x})$ que representa los individuos más prometedores de cada generación. Tales generaciones se componen de un ciclo de:

1. muestreo de $p(\vec{x})$ para obtener una población pop,
2. selección de los individuos mas prometedores pop' de pop,
3. actualización de $p(\vec{x})$ usando pop'.

Se pueden considerar diferentes EDAs dependiendo de la forma en la que se ha modelado la distribución de probabilidad $p(\vec{x})$. Para este trabajo se han considerado los siguientes:

- *Modelos de una variable*: se supone que las variables son independientes y

por lo tanto, la distribución de probabilidad $p(\vec{x})$ se factoriza de esta forma

$$p(\vec{x}) = \prod_{i=1}^{n} p(x_i).$$

El EDA más simple se denomina Algoritmo de Distribución Marginal Univariable (UMDA)[3] [259], en el cual $p(x_i)$ se calcula como

$$p(x_i = v) = \frac{1}{k} \sum_{j=1}^{k} \delta(pop'_{ji}, v),$$

donde $k = |pop'|$, pop'_{ji} es el valor de la i-ésima variable del j-ésimo individuo de pop', y $\delta(\cdot, \cdot)$ es la delta de Kronecker ($\delta(a, b) = 1$ si $a = b$ y $\delta(a, b) = 0$ en otro caso). Una generalización de UMDA es Aprendizaje Incremental Basado en Población (PBIL)[4] [30], un algoritmo en el que el modelo probabilístico se actualiza usando una combinación lineal de su valor actual y el nuevo valor aprendido durante el muestreo, es decir,

$$p'(x_i = v) = (1 - \eta)p(x_i = v) + \eta \frac{1}{k} \sum_{j=1}^{k} \delta(pop'_{ji}, v)$$

para algún parámetro de tasa de aprendizaje η ($0 < \eta \leqslant 1$) y donde el primer sumando representa el modelo actual de referencia y el segundo sumando corresponde al modelo aprendido durante esta generación. Considérese que PBIL se reduce a UMDA para $\eta = 1$.

- *Modelos de dos variables*: estos modelos pueden capturar dependencias de menor orden suponiendo relaciones entre pares de variables. Más concretamente, en los modelos considerados aquí $p(\vec{x})$ se factoriza como

$$p(\vec{x}) = p(x_{i_1}) \prod_{j=2}^{n} p(x_{i_j} | x_{i_{a(j)}}),$$

donde $i_1 \cdots i_n$ es una permutación de los índices $1 \cdots n$, y $a(j) < j$ es el índice de la permutación de la variable de la que depende x_{i_j}. Los EDAs considerados dentro de esta clase están basados en Maximización de la Información Mutua para Agrupamiento de la Entrada (MIMIC)[5] [46] y en Combinación de Optimizadores con Árboles de Información Mutua (COMIT)[6] [31]. En el primer caso, se supone que $a(j) = j - 1$ (es decir, cada variable depende de la anterior en la permutación) y la permutación se

[3] *Univariate Marginal Distribution Algorithm*
[4] *Population-Based Incremental Learning*
[5] *Mutual Information Maximization for Input Clustering*
[6] *Combining Optimizers with Mutual Information Trees*

construye eligiendo la variable i_1 con la entropía más baja $H(X_k)$ de la muestra seleccionada pop' y a continuación se elige la variable i_j $(j > 1)$ (entre aquellas que todavía no se han seleccionado) que minimiza la entropía condicional $H(X_k|X_{i_{j-1}})$.

En el segundo caso (EDA basado en COMIT), no se tiene la restricción $a(j) = j - 1$, y de este modo simplemente se selecciona la variable i_j que minimiza $H(X_k|X_{i_s}, s < j)$. Por lo tanto, mientras que en el primer caso se tiene una estructura de dependencias lineal, en este segundo caso se tiene una estructura de dependencias en forma de árbol. Considérese finalmente que en estos EDAs multimeméticos bivariables se computan de forma separada modelos para los genotipos y para los memes.

En todos los casos, la estimación de probabilidad para el modelo actualizado incluye la corrección de Laplace [57] para prevenir la convergencia prematura, siempre permitiendo una tasa de exploración superior a cero. Por otro lado, se ha considerado que cada uno de los EDAs presentados tiene una versión equivalente elitista[7], en la cual la nueva población se crea truncando la selección a partir de la unión de la muestra seleccionada en el paso anterior y la muestra extraída del modelo actual. Como se indica en [156], los EDAs elitistas con la corrección de Laplace pueden converger a una población conteniendo el óptimo global.

3.4.2. Análisis Experimental

Para analizar el rendimiento de los EDAs multimeméticos descritos en la sección previa se ha considerado un conjunto de problemas de optimización de funciones pseudobooleanas como TRAP, H-IFF, H-XOR y SAT (los cuales se describen en el Apéndice A). A continuación se analizan los resultados.

Se ha considerado que los EDAs multimeméticos tienen una población de $\mu = 128$ individuos. La selección se realiza truncando la población, de forma que siempre se mantiene al mejor 50 % de los individuos para actualizar el modelo probabilístico y la tasa de aprendizaje utilizada en PBIL es $\eta = 0{,}1$. Los memes se representan como reglas de longitud $r = 3$ y se ha considerado $w = 1$ (se realiza una reescritura y se mantiene si se mejora la solución). En todos los casos el coste de aplicar un meme se contabiliza como una fracción de la evaluación completa (como la fracción de la función de fitness que es necesario recalcular debido a los cambios producidos en el genotipo) y se añade al número de evaluaciones totales. Una ejecución termina cuando se alcanzan 50000 evaluaciones realizando 20 ejecuciones por cada problema y algoritmo. Para medir los resultados se ha incluido en la experimentación un MMA evolutivo equivalente –denominado sMMA de aquí en adelante– usando el mismo tamaño de población ($\mu = 128$) y un plan reproductivo generacional con torneo binario para la selección de los padres,

[7] Considérese que la definición original de COMIT es intrínsecamente elitista. Aquí en cambio se han considerado las dos versiones de esta aproximación, una elitista y otra no elitista

Tabla 3.4: Resultados (20 ejecuciones) de los diferentes EDAs sobre los problemas TRAP, H-IFF, H-XOR y SAT. Se muestra la mediana ($\tilde{x}$), la media ($\bar{x}$) y el error estándar de la media ($\sigma_{\bar{x}}$).

	TRAP		H-IFF	
	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$
sMMA	31.4	30.0 ± 0.5	408.0	427.6 ± 13.9
sMA	28.6	28.3 ± 0.2	370.0	363.6 ± 5.4
UMDA	20.0	20.5 ± 0.2	363.0	385.9 ± 19.5
UMDA$_e$	23.9	25.2 ± 0.8	576.0	517.5 ± 16.9
PBIL	19.2	19.2 ± 0.0	276.5	275.1 ± 3.4
PBIL$_e$	24.6	25.7 ± 0.7	441.5	441.7 ± 12.1
MIMIC	20.3	20.2 ± 0.1	328.5	330.3 ± 5.0
MIMIC$_e$	31.6	31.3 ± 0.2	472.0	493.6 ± 16.3
COMIT	21.0	21.0 ± 0.2	337.5	342.9 ± 5.9
COMIT$_e$	32.0	32.0 ± 0.0	424.0	443.8 ± 12.2
	H-XOR		SAT	
	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$
sMMA	360.0	360.2 ± 4.4	547.0	546.6 ± 0.4
sMA	356.0	355.0 ± 3.5	547.0	547.0 ± 0.4
UMDA	320.5	324.6 ± 5.2	548.0	548.0 ± 0.3
UMDA$_e$	348.0	341.6 ± 6.9	548.0	548.3 ± 0.2
PBIL	270.5	271.0 ± 2.4	543.0	543.8 ± 0.5
PBIL$_e$	259.0	258.6 ± 2.2	548.5	548.3 ± 0.3
MIMIC	310.0	312.9 ± 2.5	546.0	545.7 ± 0.4
MIMIC$_e$	393.5	397.6 ± 4.7	548.0	548.2 ± 0.2
COMIT	330.0	328.1 ± 3.7	548.0	548.0 ± 0.2
COMIT$_e$	408.0	419.4 ± 7.8	548.0	548.0 ± 0.3

operador de cruce de un punto (p_X = 1,0), mutación bit-flip ($p_M = 1/\ell$, donde $\ell = 128$ es el número de bits), búsqueda local (guiada usando el meme enlazado al individuo) y reemplazo del peor padre. En este sMMA, los descendientes heredan el meme del mejor padre, el cual está también sujeto a mutación con probabilidad p_M. Además se muestran resultados comparando el sMMA con un sMA de meme fijo en el que la búsqueda local basada en reglas se ha sustituido por una búsqueda local fija, consistente en sustituir un bit de forma aleatoria por su valor opuesto.

En la Tabla 3.4 se proporcionan resultados numéricos completos. Como se puede comprobar, si se elimina el sistema de reglas en la búsqueda local y se cambia por un mecanismo más simple el rendimiento se degrada notablemente en todos los problemas (excepto en SAT, quizá porque este es un problema lo suficientemente complejo como para que no se aprecien diferencias entre unos algoritmos y otros sin recurrir a un buen ajuste de la parametrización de los al-

Tabla 3.5: Comparación estadística entre los diferentes EDAs multimeméticos usando el test ranksum de Wilcoxon ($\alpha = 0{,}05$). Para cada problema/EDA se proporcionan tres símbolos, indicando respectivamente cómo se comporta el algoritmo con respecto a su homólogo (no-)elitista, con respecto al sMMA y con respecto al algoritmo con la mediana más alta del problema correspondiente (el cual se marca con una estrella $\star$ en tercera posición). Un círculo blanco/negro ($\circ$/$\bullet$) indica que el algoritmo etiquetado en la columna tiene una peor/mejor mediana estadísticamente significativa. Un signo '=' indica que no hay diferencia estadísticamente significativa.

	UMDA	UMDA$_e$	PBIL	PBIL$_e$	MIMIC	MIMIC$_e$	COMIT	COMIT$_e$
TRAP	$\circ\circ\circ$	$\bullet\circ\circ$	$\circ\circ\circ$	$\bullet\circ\circ$	$\circ\circ\circ$	$\bullet=\circ$	$\circ\circ\circ$	$\bullet\bullet\star$
H-IFF	$\circ\circ\circ$	$\bullet\bullet\star$	$\circ\circ\circ$	$\bullet=\circ$	$\circ\circ\circ$	$\bullet\bullet=$	$\circ\circ\circ$	$\bullet=\circ$
H-XOR	$\circ\circ\circ$	$\bullet\circ\circ$	$\bullet\circ\circ$	$\circ\circ\circ$	$\circ\circ\circ$	$\bullet\bullet\circ$	$\circ\circ\circ$	$\bullet\bullet\star$
SAT	$=\bullet=$	$=\bullet=$	$\circ\circ\circ$	$\bullet\bullet\star$	$\circ=\circ$	$\bullet\bullet=$	$=\bullet=$	$=\bullet=$

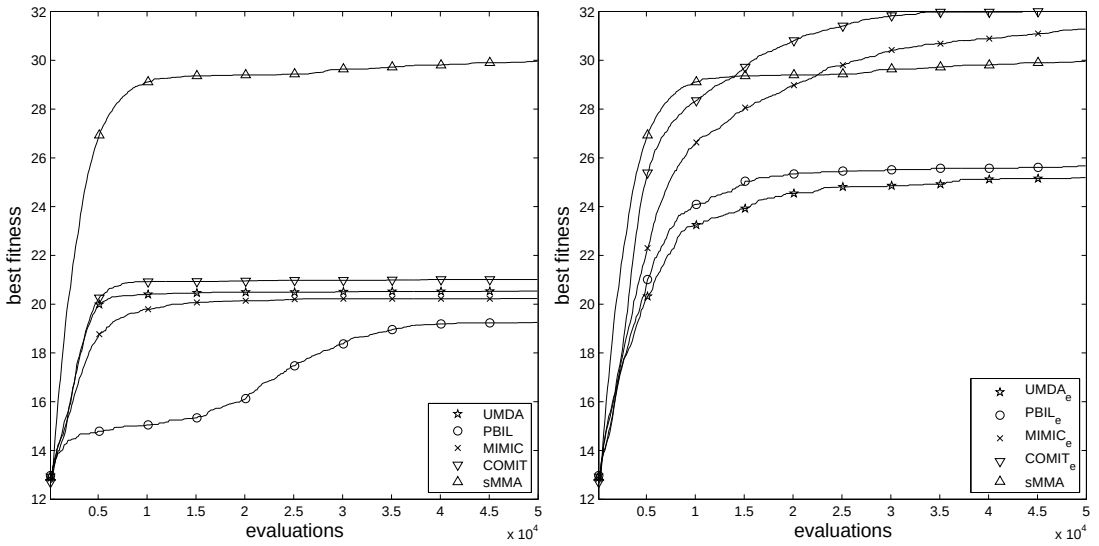


Figura 3.12: Evolución del mejor fitness en la función TRAP para EDAs multimeméticos no elitistas (izquierda) y para sus equivalentes elitistas (derecha). En ambas figuras se incluyen los resultados para el sMMA.

goritmos). En relación a los EDAs multimeméticos, considérese que las versiones elitistas de estos (denotadas con el subíndice e) tienen en general un rendimiento mucho mejor que sus equivalentes no elitistas. Por otro lado, mientras estos últimos son en muchos casos inferiores al sMMA, los EDAs multimeméticos elitistas proporcionan un alto rendimiento en todos los problemas. Todo esto se investiga más a fondo en la Tabla 3.5. Como se puede ver, la superioridad de los algoritmos elitistas sobre los no elitistas es estadísticamente significativa en todos los casos, excepto en PBIL para H-XOR y en UMDA y COMIT para SAT. Además, la superioridad de los algoritmos elitistas sobre el sMMA (símbolo central en cada

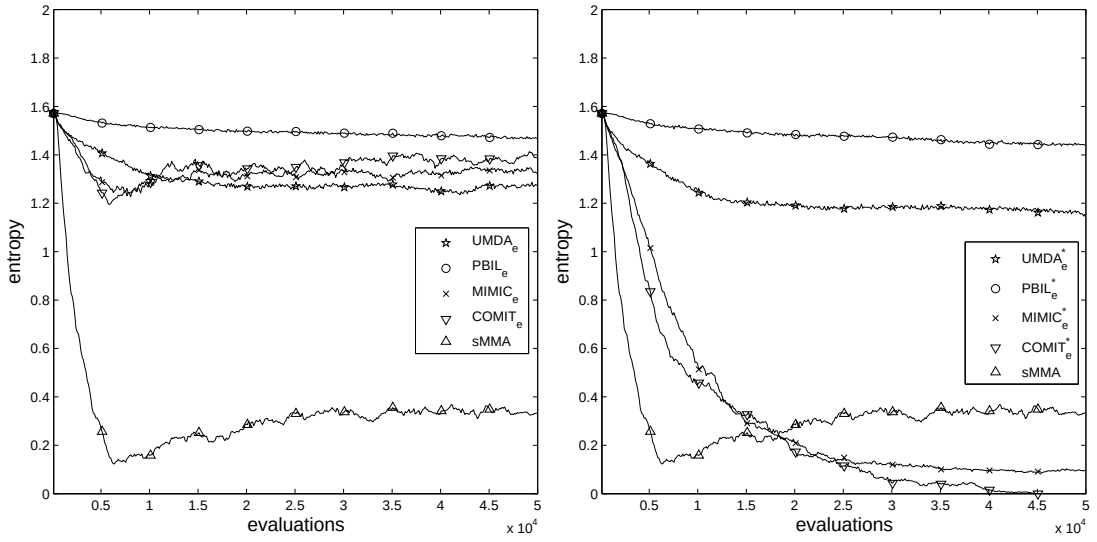


Figura 3.13: Evolución de la diversidad memética en la función TRAP para EDAs multimemé ticos usando la corrección de Laplace en el modelo probabilístico de memes (izquierda) y sin utilizar dicha corrección (derecha).

entrada de la Tabla 3.5) es también estadísticamente significativa en la mayoría de los casos. Entre los diferentes EDAs elitistas, COMIT_e parece proporcionar el mejor resultado total, siendo el mejor algoritmo en TRAP (ver Figura 3.12 para una ilustración de la evolución del fitness de este problema) y H-XOR siendo idéntico que PBIL_e en SAT. Además es interesante observar el buen rendimiento de UMDA_e para este problema. Esto puede ser debido al hecho de que la solución óptima en este caso es una cadena homogénea (todos 0s o todos 1s), una estructura que es fácilmente capturada por los memes; tan pronto como la naturaleza más simple del modelo probabilístico de UMDA dirige la búsqueda hacia un estado con predominio de cualquiera de estos símbolos, los memes pueden facilitar que se alcance este óptimo. Esta hipótesis se sostiene con los resultados comparativamente peores de los algoritmos univariable para el problema H-XOR, en el cual la solución óptima contiene un 50 %-50 % de 1s y 0s situados en localizaciones concretas (lo que hace que cualquier mitad de la solución puede ser muy diferente de la otra mitad).

Centrando ahora la atención en el efecto que tiene la corrección de Laplace sobre los algoritmos, se observa que mientras su uso es fundamental en el modelo del genotipo (los experimentos sin esta corrección indican una rápida convergencia del modelo probabilístico hacia estados subóptimos en unas pocas docenas de generaciones), a nivel memético parece mas cuestionable. Como se puede ver en la Figura 3.13 (izquierda), la entropía de los memes generados permanece muy elevada en todos los EDAs, a diferencia del sMMA que se estabiliza a un nivel más bajo de entropía. Esto indica que los EDAs están teniendo dificultades para centrar el proceso de búsqueda de los memes más adecuados, bien por las

Tabla 3.6: Resultados (20 ejecuciones) de los diferentes MMAs sobre los problemas TRAP, H-IFF, H-XOR y SAT, sin utilizar la corrección de Laplace en la estimación de las probabilidades de los memes. Se muestra la mediana ($\tilde{x}$), la media ($\bar{x}$) y el error estándar de la media ($\sigma_{\bar{x}}$).

	TRAP		H-IFF	
	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$
UMDA$^*_e$	23.8	25.2 ± 0.8	576.0	516.9 ± 17.3
PBIL$^*_e$	24.7	25.7 ± 0.7	443.0	451.5 ± 15.9
MIMIC$^*_e$	32.0	31.7 ± 0.2	464.0	473.6 ± 12.6
COMIT$^*_e$	32.0	32.0 ± 0.0	464.0	480.0 ± 13.9
	H-XOR		SAT	
	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$
UMDA$^*_e$	342.0	342.8 ± 7.6	549.0	548.4 ± 0.3
PBIL$^*_e$	262.5	259.9 ± 2.2	549.0	548.6 ± 0.3
MIMIC$^*_e$	412.0	417.0 ± 5.3	548.5	548.5 ± 0.2
COMIT$^*_e$	416.0	427.6 ± 10.0	548.0	548.4 ± 0.3

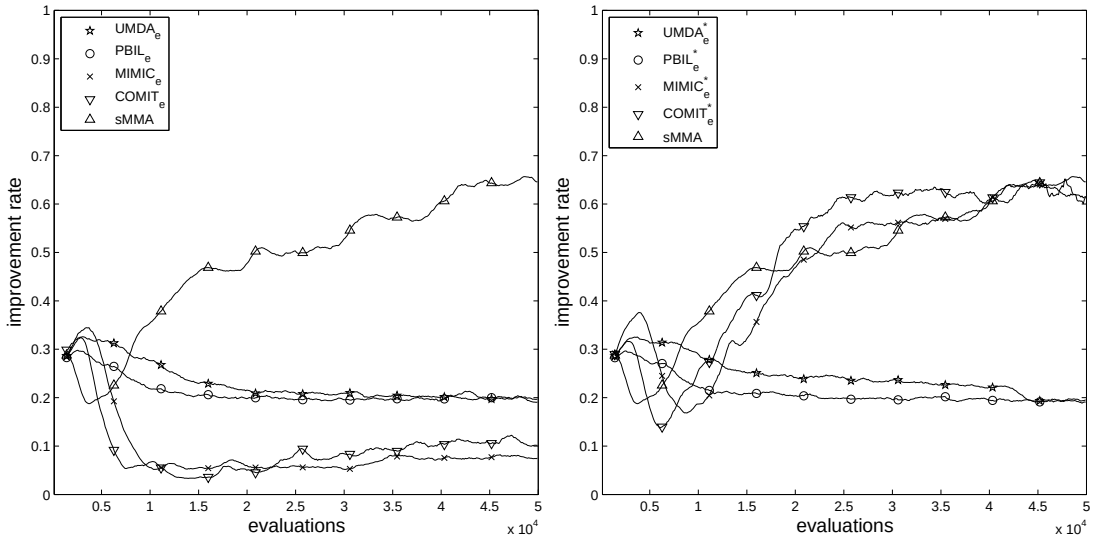


Figura 3.14: Evolución de la tasa de éxito de los memes (porcentaje de aplicaciones de memes que producen una mejora) en la función TRAP para EDAs multimeméticos usando la corrección de Laplace en el modelo probabilístico de memes (izquierda) y sin utilizar dicha corrección (derecha).

limitaciones del propio modelo probabilístico o por la perturbación exploratoria de la corrección de Laplace. Por ello, se han realizado experimentos con los EDAs elitistas pero desactivando esta corrección en el modelado de los memes. No es sorprendente que esto tenga una mayor influencia en los modelos bivariables, ya que son ahora capaces de converger a estados particulares que los modelos univariable no logran alcanzar, los cuales son incapaces de capturar la estructura

de los memes en muchos casos –ver Figura 3.13 (derecha). En general, esto tiene también una influencia positiva en el rendimiento como se muestra en la Tabla 3.6. A pesar de que los modelos univariable tienen rendimientos ligeramente mejores en algunos casos, la diferencia no es estadísticamente significativa. En el caso de los modelos bivariables, hay una diferencia estadísticamente significativa en favor de MIMIC$^*_e$ (el superíndice * denota que la corrección de Laplace se ha desactivado) para TRAP y H-XOR y en favor de COMIT$^*_e$ para H-IFF. También es interesante observar como el éxito en la aplicación de los memes (el porcentaje de aplicación de memes que producen como resultado una mejora) es mayor en estas variantes – ver Figura 3.14 – respaldando la hipótesis de que la búsqueda está más encaminada en este caso.

3.4.3. Discusión

En el contexto multimemético considerado, los EDAs parecen una aproximación bastante prometedora para este tipo de optimización memética auto-adaptativa gracias a sus ventajas bien conocidas, como son que requieren un esfuerzo considerablemente menor para ajustar la parametrización que sus equivalentes genéticos y su flexibilidad para modelar estructuras combinatorias como los memes. De hecho, la experimentación con EDAs multimeméticos ha proporcionado resultados alentadores: cuando se les dota de elitismo, los EDAs multimeméticos son claramente superiores a un MMA que manipule genes y memes mediante operadores genéticos. Asimismo se ha podido comprobar que los EDAs multimeméticos que usan modelos de probabilidad de dos variables son capaces de mejorar los resultados de los MMAs genéticos. También se ha observado que la búsqueda memética está más centrada cuando no se utiliza la corrección de Laplace en el modelado de los memes.

Capítulo 4

Algoritmos Multimeméticos basados en Islas

Como se ha comentado en el capítulo anterior la dinámica de propagación de los memes es más compleja en los MAs que en sus homólogos genéticos. Este hecho es especialmente relevante en modelos de MMAs con multipoblación, en los cuales además de las dinámicas de cada población interna hay que considerar el efecto de la comunicación entre las diferentes poblaciones. Aunque la influencia de las políticas de migración se ha estudiado con profusión en el contexto de los EAs –e.g. en [13, 53, 329]; ver también [87, 359]– no lo ha sido tanto en la familia de los MMAs. Por otra parte, hay que considerar también el papel que las políticas de migración pueden tener sobre otras propiedades del algoritmo como la diversidad de la población, ya que en esta familia de técnicas los individuos son también responsables de guiar activamente parte de la búsqueda de una forma auto-adaptativa y transportar la información con este objetivo. Por lo tanto, las decisiones de diseño involucradas en el proceso de migración no solo afectan de forma implícita al proceso de búsqueda a través de la difusión del genoma, sino que lo hacen además explícitamente por medio de la propagación de los memes.

En las siguientes secciones se analizan estas cuestiones, pero antes se describe brevemente el modelo de trabajo del MMA que se va a utilizar de aquí en adelante así como las estrategias de migración consideradas (Sección 4.1). A continuación y dada la necesidad de gestionar apropiadamente la volatilidad de los sistemas sobre los que se puede ejecutar este tipo de algoritmos, se analiza la tolerancia a fallos en el contexto de los MMAs (Sección 4.2). Finalmente se profundiza en una de las estrategias de gestión de fallos consistente en crear puntos de restauración periódicos del estado del sistema para salvaguardar la información ante inminentes caídas de los nodos (Sección 4.3).

4.1. El Modelo de Islas

El modelo de islas se basa en la utilización de una estructura espacialmente distribuida de las poblaciones naturales, es decir, la población total se divide en subpoblaciones disjuntas ejecutándose en cada una de ellas un MMA clásico. Para mitigar el aislamiento propio de las poblaciones separadas y evitar que convergan a óptimos locales se realizan intercambios regulares de individuos entre las distintas subpoblaciones (inyectando de esta manera nuevos individuos y por lo tanto mejorando la diversidad) según un criterio definido previamente y de una manera establecida según diversos parámetros involucrados en el proceso completo que afectan al mecanismo de selección de los migrantes, a la forma en la que los migrantes reemplazan a los individuos de la población original o al número de individuos que forman parte del proceso de migración cada vez.

4.1.1. Descripción del Modelo

El modelo básico subyacente es el descrito en la Sección 3.1, con un MMA que trata explícitamente los memes a través del uso de reglas de reescritura adheridas a los individuos. Los padres se seleccionan por torneo binario, con operaciones de recombinación, mutación y búsqueda local mediante el meme asociado a cada individuo y una política de reemplazo en la que los peores padres se sustituyen por la descendencia.

En este caso, el MMA (iMMA de ahora en adelante) se ha implementado sobre una arquitectura basada en islas, a través de la cual múltiples poblaciones independientes (islas) se crean inicialmente para a continuación evolucionar de forma parcialmente aislada –considérese que en cada isla se ejecuta el mismo MMA con la misma parametrización, aunque esta restricción podría ser alterada y ejecutar algoritmos con configuraciones diferentes en cada isla o incluso algoritmos completamente diferentes –e.g., en algunos nodos se podría ejecutar un GA y en otros un MA. La definición de esta arquitectura conlleva algunas decisiones de diseño relacionadas con:

1. *La topología de interconexión entre las islas*: por ejemplo, un anillo, una rejilla, un hipercubo, etc.

2. *La política de migración*: se trata de una colección de paramétros de migración que determinan cómo y cuándo se intercambia la información entre las islas [13, 53, 329].

Para implementar el MMA descrito previamente sobre un modelo multi-isla es necesario definir las dos cuestiones anteriores. Con respecto a la topología de interconexión se va a utilizar inicialmente un anillo unidireccional, por ser una topología comúnmente empleada y además lo suficientemente simple para poder centrar el estudio en otros aspectos como las estrategias de migración. Por otro lado, la política de migración abarca determinados parámetros como el número

m de individuos que se someten al proceso migratorio, la frecuencia ζ de tales eventos, el procedimiento ω_S utilizado para seleccionar los individuos que se van a migrar desde la isla emisora, el procedimiento ω_R empleado para gestionar los migrantes en la isla receptora y si el carácter de la interacción es síncrono o asíncrono –ver [13]. En esta sección se va a considerar que la interacción es síncrona y lo relativo a ω_S y ω_R se especificará a continuación. Nótese en este sentido que la naturaleza de estos parámetros es cualitativa, en contraposición a la naturaleza cuantitativa de los parámetros m y ζ. Por este motivo, su estudio no puede por lo tanto realizarse utilizando un ajuste meramente numérico.

4.1.2. Estrategias de Migración

En esta sección se profundiza en diversas estrategias de migración que se pueden implementar en un MMA basado en islas para analizar posteriormente en la Sección 4.1.3 el impacto que tiene la elección de las estrategias de migración sobre el rendimiento del algoritmo. Con respecto al operador de selección de migrantes ω_S, se han considerado las siguientes seis posibilidades:

- best: los mejores m individuos de la población emisora se seleccionan para la migración. Esta estrategia se podría ver como un intento de proporcionar el empuje máximo inmediato en el fitness de la isla receptora, probablemente inducido porque si los migrantes empiezan a dominar la población se produciría un reenfoque de la búsqueda.

- random: los migrantes se seleccionan por muestreo aleatorio (sin reemplazo) en la población emisora. En este caso, se trata de inyectar diversidad en la población objetivo a través de un muestreo aleatorio del material genético/memético de la isla emisora.

- probabilistic: esta estrategia utiliza las ideas de los algoritmos de estimación de distribuciones introducidas en el capítulo precedente y debe entenderse dentro del espíritu de la estrategia anterior. Aquí se crea un modelo probabilístico de la población emisora que se utiliza para producir los migrantes. Por lo tanto, proporciona una muestra de la información contenida en la isla emisora pero no necesariamente correspondiendo a los individuos exactos existentes. De este modo, se puede ver como una selección más exploratoria que aleatoria. En este caso se ha considerado un modelo univariable simple en el que los migrantes se generan a partir de la probabilidad de que cada símbolo en una posición dada encaje con la frecuencia relativa de cada símbolo en esa misma posición entre la población.

- diverse-gene: en la línea del algoritmo multikulty [22], los migrantes se seleccionan para introducir tanta diversidad como sea posible en la población destino, cf. [63]. Para este fin, se seleccionan los individuos cuya distancia genotípica (en el sentido de Hamming) a los individuos de la población receptora sea máxima.

- diverse-meme: esta es la extensión natural de la estrategia previa en el campo memético. En este caso, los migrantes son los individuos que transportan los memes cuya distancia (de nuevo en sentido de Hamming) a los memes en la isla receptora sea máxima. El objetivo no es, por lo tanto, introducir diversidad genética de forma explícita sino hacer esto implícitamente, introduciendo diversidad en cuanto a la forma en la que las soluciones se mejoran.

- random-immigrants: esta estrategia genera los migrantes de forma totalmente aleatoria [162]. Ya que esto no tiene en cuenta la isla emisora en absoluto, esta estrategia representa un intento de medir el efecto bruto de introducir nuevos individuos en la población destino, desacoplándolo del efecto atribuido al intercambio de información entre islas. De alguna forma, esto proporciona una referencia en cuanto al rendimiento por encima del cual las otras estrategias deberían situarse.

Con respecto al operador de reemplazo de los migrantes ω_R, se han considerado las siguientes dos posibilidades:

- replace-worst: se reemplazan los peores individuos de la población por los migrantes que llegan.

- replace-random: los migrantes reemplazan a los individuos seleccionados de forma aleatoria.

En cualquier caso, se ha elegido el funcionamiento del reemplazo de forma totalmente incondicional (es decir, los migrantes siempre se aceptan por la población destino) por dos razones: en primer lugar, se pretende maximizar el efecto (positivo o negativo) de la operación de migración y en segundo lugar, se intenta promover la diversidad sobre la posible pérdida inmediata de fitness (recalcar que en los MMAs, las soluciones están sujetas a mejora local y por lo tanto, tal pérdida se podría mitigar a través de la aplicación del meme; más incluso explorando el hecho de que la atracción hacia otro óptimo podría ser una ventaja más sólida que una buena solución temporal).

4.1.3. Análisis Experimental

Las estrategias de migración introducidas en la sección previa se han examinado de forma experimental con bastante profundidad. Antes de presentar los resultados, se describe la configuración de los experimentos y el conjunto de problemas de test que se han utilizado en la experimentación: TRAP, H-IFF, H-XOR y SAT (ver Apéndice A para más detalles).

Se ha considerado un MMA basado en islas tal como se ha descrito en la Sección 4.1.1, con un tamaño de población de $\mu = 128$ individuos, una probabilidad de recombinación $p_X = 1{,}0$ y una probabilidad de mutación $p_M = 1/\ell$ ($\ell = 128$, es la longitud del genoma). Esta población se coloca entre las $n_\iota \in 1, 2, 4, 8$ islas,

cada una de ellas compuesta por μ/n_ι individuos. El caso $n_\iota = 1$ (denominado como sMMA) corresponde a panmixia y no tiene migración alguna. En el resto de escenarios, las islas se colocan en un anillo unidireccional y la migración tiene lugar cada $\zeta = 20$ generaciones, por lo tanto, permitiendo un lapso suficiente de evolución aislada de cada deme, cf. [13]. Se selecciona un migrante usando ω_S y se inserta en la población receptora utilizando ω_R, donde ambos, ω_S y ω_R son las estrategias descritas en la Sección 4.1.2. Los memes se expresan como reglas de longitud $r = 3$ y se considera $w = 1$. En todos los casos el coste de aplicar un meme se contabiliza como una fracción de la evaluación (es decir, como una fracción de la función de fitness que necesita reevaluarse como resultado de un cambio en el genotipo) y se añade al número total de evaluaciones. Cada ejecución termina cuando se alcanzan 50000 evaluaciones y se ejecutan 20 ejecuciones para cada combinación de problema, número de islas, ω_S y ω_R.

En primer lugar, se proporciona un conjunto completo de resultados numéricos en la Tabla 4.1. Como es de esperar, la calidad de los resultados mejora globalmente cuando el número de islas crece (considérese que todos los problemas utilizados son problemas de maximización, y por lo tanto, los valores más elevados son los mejores). Esta es una consecuencia bien conocida del uso de los EAs descentralizados que se manifiesta de forma natural en el contexto multimemético. El principal punto de interés del análisis no está en cualquier caso en cómo se logran los mejores resultados, lo que puede obtenerse fácilmente incrementado el numero de islas (esto es también interesante y podría ser objeto de una investigación posterior), sino en el efecto que las decisiones de diseño relativas a la selección de los migrantes y al reemplazo de estos pueda tener en el rendimiento del algoritmo. Por ello, se ha realizado un análisis estadístico sistemático para determinar el impacto relativo que cada política de migración tiene sobre el iMMA.

Considérense inicialmente todas las estrategias de migración tanto para $\omega_R =$ replace-worst como para $\omega_R =$ replace-random. Se realiza una comparación basada en ranking para calcular el orden relativo de cada operador ω_S para un problema dado y un número de islas n_ι determinado: la estrategia de selección con la mejor media es elegida con el ranking 1 y la peor con el ranking 6 (recalcar que hay seis operadores ω_S). En caso de empates se considera el promedio de las posiciones empatadas. La Figura 4.1 muestra la distribución de los rankings para cada operador ω_S. Puede observarse que estos rankings son en su mayor parte consistentes para ambas estrategias de reemplazo de los migrantes. El hecho de que random-immigrants se clasifique habitualmente en las últimas posiciones es compatible con que el iMMA se esté beneficiando de la información intercambiada entre las islas más allá de una pura diversidad aleatoria (y además consigue mejores resultados conforme aumenta el número de islas –ver Tabla 4.1). Además, best se clasifica en una pobre posición en ambos casos. Esto a menudo se atribuye a una convergencia prematura inducida por esta estrategia más intensiva. Este efecto se ilustra en la Figura 4.2 (a). En cualquier caso, se observa que una diversidad global más elevada no se traduce per se en un mejor rendimiento: los

Tabla 4.1: Resultados (20 ejecuciones) de los diferentes iMMAs sobre los problemas TRAP, H-IFF, H-XOR y SAT, usando la estrategia replace-worst (mitad superior) y la estrategia replace-random (mitad inferior). Se muestran los resultados de la mediana ($\tilde{x}$), la media ($\bar{x}$) y el error estándar de la media ($\sigma_{\bar{x}}$).

		TRAP		H-IFF		H-XOR		SAT	
replace-worst		$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$
sMMA	$n_i = 1$	31.4	30.0 ± 0.5	408.0	427.6 ± 13.9	360.0	360.2 ± 4.4	547.0	546.6 ± 0.4
random	$n_i = 2$	30.6	30.3 ± 0.3	456.0	471.1 ± 17.2	380.0	382.4 ± 6.2	547.0	547.3 ± 0.3
	$n_i = 4$	31.2	30.6 ± 0.4	520.0	509.2 ± 15.7	408.0	416.0 ± 10.5	548.0	547.5 ± 0.4
	$n_i = 8$	31.6	30.7 ± 0.4	576.0	564.0 ± 8.3	412.0	415.1 ± 7.3	547.0	547.5 ± 0.3
best	$n_i = 2$	29.6	30.0 ± 0.3	436.0	470.0 ± 20.4	372.0	379.4 ± 7.6	547.0	547.1 ± 0.4
	$n_i = 4$	30.8	30.3 ± 0.4	456.0	469.4 ± 17.2	380.0	385.8 ± 5.4	546.0	546.5 ± 0.4
	$n_i = 8$	31.4	30.3 ± 0.4	576.0	519.6 ± 14.5	374.0	378.2 ± 6.9	547.0	546.9 ± 0.4
diverse-gene	$n_i = 2$	29.6	29.6 ± 0.4	456.0	473.8 ± 16.5	384.0	384.0 ± 4.7	548.0	547.4 ± 0.3
	$n_i = 4$	30.2	30.0 ± 0.4	528.0	499.7 ± 18.3	395.0	404.6 ± 10.7	547.0	546.9 ± 0.4
	$n_i = 8$	30.4	30.1 ± 0.4	576.0	543.6 ± 11.8	394.0	404.8 ± 8.5	547.0	547.3 ± 0.4
diverse-meme	$n_i = 2$	30.6	30.1 ± 0.4	456.0	475.8 ± 16.0	380.0	381.4 ± 4.4	548.0	547.4 ± 0.4
	$n_i = 4$	31.0	30.5 ± 0.4	472.0	501.2 ± 16.1	404.0	411.2 ± 11.4	548.0	547.5 ± 0.3
	$n_i = 8$	31.2	30.7 ± 0.3	576.0	553.2 ± 10.5	402.0	418.2 ± 11.9	548.0	547.5 ± 0.3
random-immigrants	$n_i = 2$	29.4	29.4 ± 0.5	436.0	451.0 ± 15.7	352.0	356.4 ± 5.0	547.0	547.5 ± 0.3
	$n_i = 4$	28.6	28.7 ± 0.5	454.0	453.2 ± 16.0	348.0	351.6 ± 3.6	547.0	546.9 ± 0.3
	$n_i = 8$	30.4	29.3 ± 0.5	454.0	471.1 ± 14.8	331.0	336.9 ± 4.3	547.0	546.8 ± 0.3
probabilistic	$n_i = 2$	31.4	30.5 ± 0.4	456.0	493.6 ± 17.6	374.0	386.2 ± 7.2	547.0	547.0 ± 0.3
	$n_i = 4$	32.0	30.8 ± 0.4	464.0	500.0 ± 13.6	394.0	387.6 ± 5.0	548.0	547.4 ± 0.3
	$n_i = 8$	32.0	30.4 ± 0.5	576.0	551.6 ± 11.3	390.0	390.2 ± 3.9	547.0	547.4 ± 0.3
		TRAP		H-IFF		H-XOR		SAT	
replace-random		$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$
sMMA	$n_i = 1$	31.4	30.0 ± 0.5	408.0	427.6 ± 13.9	360.0	360.2 ± 4.4	547.0	546.6 ± 0.4
random	$n_i = 2$	30.4	30.1 ± 0.3	456.0	480.0 ± 18.9	380.0	390.4 ± 7.5	547.0	547.0 ± 0.4
	$n_i = 4$	30.8	30.2 ± 0.4	520.0	506.8 ± 16.5	412.0	426.6 ± 10.0	547.0	547.2 ± 0.3
	$n_i = 8$	31.4	30.7 ± 0.3	576.0	543.2 ± 11.6	408.0	414.9 ± 6.4	548.0	547.5 ± 0.3
best	$n_i = 2$	29.6	30.0 ± 0.4	440.0	457.2 ± 19.3	378.0	379.2 ± 5.8	547.0	547.1 ± 0.3
	$n_i = 4$	31.0	30.3 ± 0.4	468.0	482.8 ± 20.5	378.0	390.6 ± 7.8	547.5	547.0 ± 0.5
	$n_i = 8$	31.4	30.2 ± 0.5	576.0	518.8 ± 14.8	384.5	384.6 ± 6.6	547.0	547.1 ± 0.3
diverse-gene	$n_i = 2$	30.6	30.2 ± 0.4	456.0	471.0 ± 17.4	384.0	385.7 ± 8.1	547.0	546.7 ± 0.3
	$n_i = 4$	30.0	29.7 ± 0.4	520.0	503.8 ± 17.1	385.0	391.9 ± 7.7	547.0	546.8 ± 0.4
	$n_i = 8$	31.6	30.4 ± 0.4	576.0	546.8 ± 11.6	382.0	388.5 ± 5.7	547.0	546.6 ± 0.3
diverse-meme	$n_i = 2$	31.0	30.2 ± 0.4	464.0	490.6 ± 18.6	388.0	386.0 ± 6.6	547.0	547.2 ± 0.3
	$n_i = 4$	30.6	30.5 ± 0.3	472.0	503.2 ± 14.1	396.0	405.1 ± 5.9	548.0	547.6 ± 0.3
	$n_i = 8$	32.0	31.1 ± 0.3	576.0	535.0 ± 12.8	408.0	411.9 ± 9.4	547.0	547.3 ± 0.3
random-immigrants	$n_i = 2$	30.6	29.8 ± 0.5	432.0	442.4 ± 15.1	352.0	355.6 ± 3.9	546.5	546.9 ± 0.4
	$n_i = 4$	28.4	28.8 ± 0.5	456.0	446.3 ± 19.6	352.0	353.9 ± 2.9	548.0	547.3 ± 0.4
	$n_i = 8$	28.6	29.1 ± 0.5	456.0	475.8 ± 16.4	338.0	337.8 ± 4.1	547.0	547.0 ± 0.4
probabilistic	$n_i = 2$	30.9	30.5 ± 0.4	464.0	482.4 ± 16.8	370.0	372.4 ± 5.1	547.0	546.7 ± 0.4
	$n_i = 4$	31.6	30.9 ± 0.3	576.0	518.6 ± 15.5	384.0	384.6 ± 4.9	547.0	547.3 ± 0.3
	$n_i = 8$	31.0	30.3 ± 0.5	576.0	548.4 ± 13.1	394.0	400.9 ± 5.8	547.0	547.3 ± 0.3

MMAs también requieren que los memes soporten de forma sostenida el proceso de búsqueda y estrategias tales como random y diverse-meme son mejores en este aspecto –ver Figura 4.2 (b). De hecho, otra observación interesante es que las estrategias de selección de migrantes basadas en diversidad memética funcionan mejor que sus homólogas genéticas. Esto indica que la inyección de nuevo material memético diferente puede tener una mayor influencia en el comportamiento del algoritmo que introducir solo material genético, en la línea del papel activo que tienen los memes actualmente en el proceso de búsqueda. La estrategia random proporciona un alto rendimiento también, lo que podría atribuirse a que constituye una solución intermedia entre las estrategias de diversidad genética y

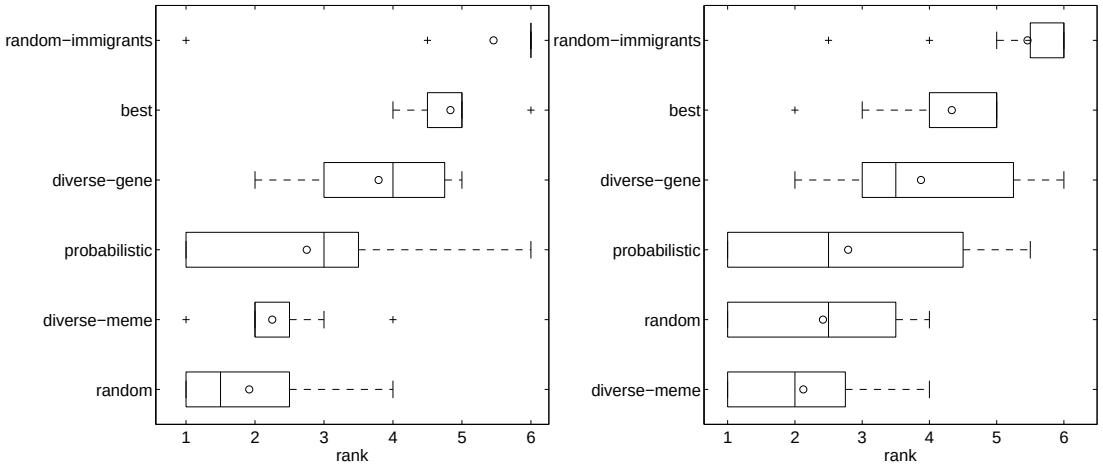
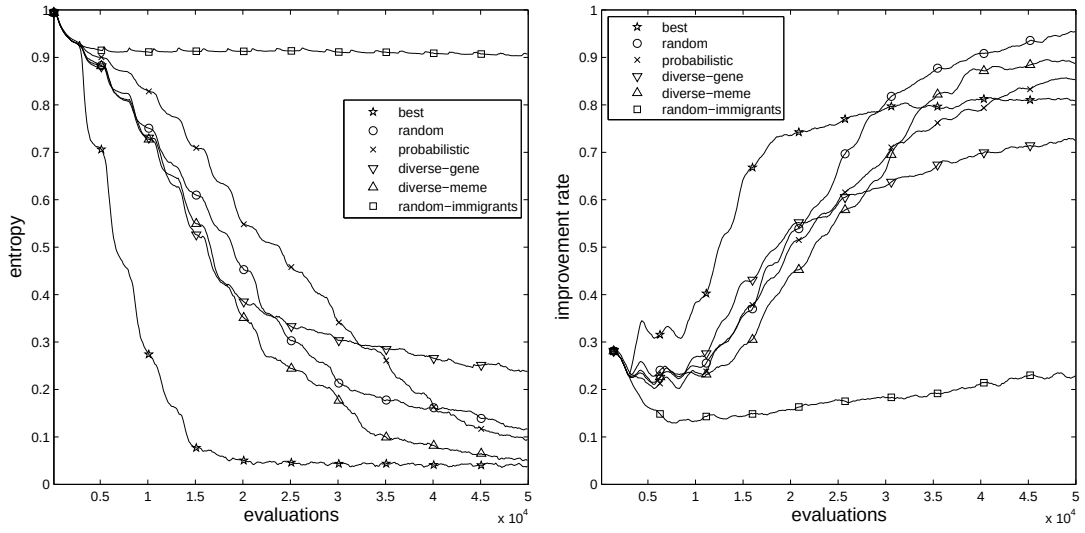


Figura 4.1: Distribución de rankings de las estrategias de selección de migrantes. Cada caja se compone del primer al tercer cuartil de la distribución, de la mediana (cuartil 2) que se identifica mediante una línea vertical, de la media representada con un círculo y de los datos con valores extremos que se indican con un signo más. Las líneas horizontales tienen una amplitud de 1.5 veces el rango entre cuartiles. (Izquierda) Resultados para $\omega_R =$ replace-worst. (Derecha) Resultados para $\omega_R =$ replace-random.

memética. Como puede verse, esta estrategia de selección produce también ligeras mejoras cuando se usa conjuntamente con replace-worst con respecto a cuando se usa con replace-random debido a que su naturaleza eminentemente exploratoria es compensada por el carácter más intensivo de la estrategia de reemplazo. Finalmente, es interesante recalcar cómo la generación probabilística de migrantes se sitúa confortablemente en la tercera posición en todos los casos, no demasiado lejos de la selección random. Obviamente, un modelo univariable simple no puede capturar adecuadamente las interdependencias entre las variables y por lo tanto esta estrategia tiene un comportamiento que puede considerarse una variante exploratoria de la estrategia random.

Para determinar el alcance a partir del cual las diferencias en el ranking son significativas se han utilizado dos tests estadísticos no paramétricos bien conocidos, los denominados Tests de Friedman [129–131] y de Quade [302]. Los resultados en el nivel estándar de $\alpha = 0{,}05$ se muestran en la Tabla 4.2. Los valores estadísticos obtenidos son claramente más altos que los valores críticos, de modo que esto indica que hay diferencias significativas en sus rankings. Por otro lado, se ha realizado adicionalmente un análisis post-hoc usando el test de Holm [175] para determinar si las diferencias son significativas con respecto a una estrategia de control (en este caso la estrategia que produce el mejor ranking medio como se muestra en la Figura 4.1). La Tabla 4.3 muestra los resultados de este test. Considérese también que el test lo superan en todos los casos las estrategias random-immigrants, best y diverse-gene y que, por consiguiente, el algoritmo



(a) (b)

Figura 4.2: (a) Entropía de la población global. (b) Tasa de éxito de la aplicación de memes (porcentaje de aplicaciones de memes que producen una mejora en el fitness del individuo). En ambos casos los datos corresponden al problema H-IFF, $n_\iota = 8$, $\omega_R =$ replace-worst.

Tabla 4.2: Test estadístico para las estrategias de selección de migrantes ($\alpha =$ 0,05).

	Valor crítico	replace-worst	replace-random
Friedman	11.070498	35.416667	28.238095
Quade	2.382823	14.811603	9.189948

de control es significativamente mejor que estos. En cambio, no hay diferencias estadísticas significativas entre diverse-meme, random y probabilistic.

Si se realiza un análisis a través de la dimensión de reemplazo, es decir, manteniendo fija la estrategia de selección y comparando ambas estrategias de reemplazo, se observa que replace-worst funciona algo mejor que replace-random. Sin embargo, en ningún caso la diferencia llega a alcanzar el nivel significativo de 0,05 usando el test de ranksum de Wilcoxon para realizar las comparaciones uno a uno entre ambas estrategias de reemplazo para cada combinación (problema, ω_S, n_ι). Si se analiza de forma específica el par (ω_S, ω_R), se puede encontrar que existen diferencias estadísticamente significativas en la distribución de ranking de las 12 combinaciones (usando los test de Friedman y Quade se obtienen los valores 71.317308 y 11.251898 respectivamente, mucho mayores que sus valores críticos correspondientes que son 19.675138 y 1.868615). El test de Holm también se ha ejecutado a continuación, como se muestra en la Tabla 4.4, con resultados consistentes con los obtenidos previamente; el test lo superan todos los pares involucrados random-immigrants, best y diverse-gene usando random+replace-worst

Tabla 4.3: Resultados del Test de Holm. (Arriba) ω_R = replace-worst usando ω_S = random como estrategia de control. (Abajo) ω_R = replace-random usando ω_S = diverse-meme como estrategia de control.

	i	Estrategia	z-statistic	p-valor	α/i
replace-worst	1	diverse-meme	0.436436	0.331260	0.050000
	2	probabilistic	1.091089	0.137617	0.025000
	3	diverse-gene	2.454951	0.007045	0.016667
	4	best	3.818813	0.000067	0.012500
	5	random-immigrants	4.637130	0.000002	0.010000
replace-random	1	random	0.381881	0.351275	0.050000
	2	probabilistic	0.872872	0.191367	0.025000
	3	diverse-gene	2.291288	0.010973	0.016667
	4	best	2.891387	0.001918	0.012500
	5	random-immigrants	4.364358	0.000006	0.010000

Tabla 4.4: Resultados del Test de Holm para todas las combinaciones de estrategias de selección/reemplazo usando random+replace-worst como estrategia de control.

i	Estrategia	z-statistic	p-valor	α/i
1	diverse-meme+replace-worst	0.651059	0.257504	0.050000
2	diverse-meme+replace-random	0.735980	0.230871	0.025000
3	probabilistic+replace-worst	1.075663	0.141039	0.016667
4	random+replace-random	1.103970	0.134803	0.012500
5	probabilistic+replace-random	1.755029	0.039627	0.010000
6	diverse-gene+replace-worst	2.745772	0.003018	0.008333
7	diverse-gene+replace-random	3.085455	0.001016	0.007143
8	best+replace-random	3.623287	0.000145	0.006250
9	best+replace-worst	4.132811	0.000018	0.005556
10	random-immigrants+replace-worst	5.123554	0.000000	0.005000
11	random-immigrants+replace-random	5.180167	0.000000	0.004545

como algoritmo de control. No se han encontrado diferencias significativas entre los pares relacionados con diverse-meme, random y probabilistic.

4.1.4. Discusión

La elección de los operadores de selección de migrantes y de reemplazo de migrantes tiene un impacto crucial en el rendimiento de los EAs basados en islas como es bien conocido. No obstante, los resultados indican que la elección del operador de selección de los migrantes es más importante que la estrategia de reemplazo. Además, se han podido confirmar algunos resultados que previamente se han ido presentando en el contexto de los GAs, tales como el hecho de que una migración basada en el mejor individuo conduce a una rápida degradación de la diversidad y disminuye su rendimiento. Por otro lado, se ha llegado a la conclusión de que una estrategia de selección puramente enfocada al mantenimiento

de la diversidad del genotipo no puede compararse favorablemente al resto de estrategias basadas en diversidad memética (aunque la primera todavía es capaz de proporcionar mejores resultados que un modelo panmíctico de una sola población o que una estrategia basada en inmigrantes aleatorios). Por otro lado, estas últimas tienen un funcionamiento estadísticamente similar a las dos estrategias consideradas con muestreo aleatorio de la población emisora. Es también especialmente interesante considerar que el modelo probabilístico de la población (incluso el más simple como el modelo univariable considerado aquí) es todavía competitivo con algunos de los operadores de migración.

4.2. Tolerancia a Fallos

El uso de modelos de computación evolutiva paralelos y distribuidos (EAs) está muy extendido en la actualidad debido a su utilidad para mejorar la calidad de las soluciones encontradas y reducir el coste computacional cuando se está tratando con problemas cuya optimización es compleja. Para este propósito, los entornos de computación emergentes, tales como las redes P2P y los *desktop grids* están ofreciendo una miríada de nuevas oportunidades (ver Sección 2.6.2.5 para más detalles sobre las plataformas de computación para los EAs distribuidos), pero también proporcionando nuevos retos: el funcionamiento sobre una red de computación cuyos recursos son volátiles hace necesaria una implementación tolerante a fallos y resistente a los cambios. En esta sección se analizan estas cuestiones desde el punto de vista de los EAs multipoblación (EAs basados en islas). Se van a considerar dos variantes diferentes de EAs, como son los GAs y los MAs, y se analizará el impacto sobre el rendimiento que tienen las decisiones de diseño en función de la topología lógica de interconexión entre las diferentes islas y la política de gestión de fallos empleada en cada caso.

4.2.1. Necesidad

La última década ha sido testigo de entornos emergentes masivamente paralelos, tales como las redes P2P [251] y las redes de computación voluntarias (*volunteer computing networks*) [320], ofreciendo grandes posibilidades para todas estas técnicas y también nuevos retos. Entre estos últimos, se puede citar la inherente volatilidad de los recursos computacionales en estas redes. El término *churn* se ha acuñado para denotar el efecto colectivo de una plétora de pares entrando o saliendo del sistema a lo largo del tiempo. Un EA paralelo ejecutándose sobre tal entorno tiene que ser capaz de tratar con este efecto, ya que el funcionamiento ideal del algoritmo mostrado en un entorno estable es perturbado por fallos de comunicaciones entre nodos o por pérdida de información debido a que puede haber en un momento dado nodos de computación que salen del sistema. Afortunadamente, los EAs han demostrado que son bastante resistentes a este fenómeno en entornos de granularidad fina, como por ejemplo los modelos

maestro-esclavo, en los que un nodo fiable se ocupa de cómo se distribuyen en el sistema tanto la ejecución del algoritmo como las operaciones sobre los individuos (p. ej. evaluación, aplicación de los operadores, etc.) o incluso cuáles pueden ejecutarse de una forma totalmente descentralizada –ver [214, 226–228]– lo que sugiere que estas técnicas son inherentemente tolerantes a fallos a este nivel. Se han utilizado diferentes políticas de gestión de fallos en este contexto de granularidad fina, tales como la redundancia [134] o los algoritmos epidémicos [135], por citar algunas referencias como recapitulación de estas aproximaciones, así como entornos de trabajo que las soportan –ver en [172]. Este escenario admite diferentes perspectivas cuando se considera un nivel de información más amplio, en particular, si se considera la distribución de subconjuntos de islas o poblaciones en el sistema, lo que ha sido abordado por ejemplo en [238] donde se propone una política de checkpointing (en la que se graba periódicamente una instantánea del estado de todas las tareas para evitar tener que reanudar la computación desde el principio). Más concretamente, proponen una arquitectura mixta en la que las islas se ejecutan en un nodo cliente estable y las operaciones individuales más costosas, tales como la búsqueda local, se realizan en nodos volátiles. Además, enfatizan la necesidad de una gestión efectiva de las cuestiones involucradas con la tolerancia a fallos. Un punto de vista relacionado con lo anterior se puede encontrar en [172], donde se estudia la robustez de un EA basado en islas con topología de anillo: cuando tiene lugar un fallo se desconecta una isla completa del anillo y este se reconecta para cerrar el hueco. Incluso con una política de gestión de fallos tan simple como esta, el EA es capaz de conseguir tan solo una pequeña degradación del rendimiento, proporcionando resultados de similar calidad a un EA sin fallos.

4.2.2. Modelos del Entorno Computacional

Uno de los objetivos de esta sección es proporcionar un conjunto de resultados sobre tolerancia a fallos de dos EAs diferentes. En particular, se consideran tanto GAs como MMAs. En relación a los últimos se definen como en la Sección 3.1. En cuanto a los primeros, considérese que si se deshabilita la evolución de los memes y el uso de la búsqueda local el MMA anterior se reduce a un GA. Ambos EAs se han implementado sobre una arquitectura basada en islas (ver Sección 4.1) con diferentes topologías de interconexión:

- *Anillo*: las islas se organizan en un anillo lógico de forma que cada isla i se puede comunicar con otras dos islas $i-1$ y $i+1$ (todas las operaciones se realizan módulo el número de islas para mantener cerrado el anillo). Considérese en este caso que se trata de un anillo bidireccional, a diferencia del utilizado en la Sección 4.1.

- *Von Neumann* (VN): las islas se colocan en una rejilla toroidal $p \times q$, donde las conexiones se realizan entre cada isla y aquellas otras que están localizadas a una distancia de Manhattan de ρ (el radio del vecindario) como

máximo. Para $\rho = 1$ esto implica que cada isla se comunica únicamente con aquellas otras situadas al norte, sur, este y oeste de su posición.

- *Hipercubo* (HC): las islas se colocan sobre los vértices de un hipercubo κ-dimensional (por lo tanto, se asume que deben existir 2^κ islas). Las conexiones se realizan a lo largo de los ejes del hipercubo, concretamente entre los vértices cuyos índices expresados en binario difieren en únicamente un bit.

- *Escala Libre* (SF): esta es una topología compleja e irregular observada habitualmente en multitud de procesos naturales y sociales, en la cual el grado de los nodos sigue una distribución potencial. Este tipo de topología se genera usando el modelo de Barabási-Albert [312] (ver Algoritmo 5), según el cual, una red crece a través de la adición de un nuevo nodo cada vez. Este nodo se conecta a otros m nodos existentes, seleccionados con una probabilidad proporcional a su grado actual (el modelo viene de esta forma conducido por procesos de conexión preferente[1] [34]).

En la Fig. 4.3 se ilustran las topologías mencionadas anteriormente. Con respecto a la estrategia de migración, se han utilizado los resultados previos obtenidos tanto en la Sección 4.1 como en la literatura del área ([13]) en el contexto de los GAs y de los MMAs, los cuales indican que la selección como migrante de un individuo de forma aleatoria y el posterior reemplazo del peor individuo de la población (isla) destino es una estrategia lo suficientemente robusta. Considérese finalmente que la migración se realiza de forma síncrona entre las islas activas. Aunque esto podría no corresponderse con el funcionamiento de muchos entornos reales es más sencillo, lo que permite realizar un análisis más directo sobre las estrategias de tolerancia a fallos y sobre las topologías subyacentes.

El modelo basado en islas descrito en las secciones anteriores se ha implementado sobre un sistema distribuido simulado compuesto por n_ι nodos cuya disponibilidad va cambiando dinámicamente. Más concretamente, se asume que los n_ι nodos están inicialmente disponibles, pero cualquiera de ellos puede abandonar el sistema durante la ejecución (es decir, puede fallar), probablemente volviendo a estar activo posteriormente (es decir, recuperándose del fallo) o quizá no volviendo a estar disponible nunca más. Existen diversas posibilidades para modelar estos fallos/recuperaciones. La aproximación más sencilla es considerar que los fallos (y las recuperaciones) son independientes tanto de los otros nodos como del tiempo, sucediendo con una cierta probabilidad p_f por unidad de tiempo. En este caso, los tiempos de disponibilidad se distribuyen exponencialmente. En una situación mas general (y realista), los fallos/recuperaciones no son resultado de un proceso sin memoria –es decir, independientes del tiempo– y por lo tanto, los tiempos de disponibilidad siguen una distribución diferente, tal como por ejemplo la distribución de Weibull [360]. Esta distribución se utiliza a menudo en el

[1] *preferential attachment*

Algoritmo 5: Modelo de Barabási-Albert

```
function BA-Model (↓ m, n : ℕ) returns net : Network
// net: La red creada
// n: Número de nodos
// m: Número de enlaces de cada nodo
m_0 ← mín(n, m);
net ← CrearClique(m_0);
δ[1 ... m_0] ← m_0;
for i ← m_0 + 1 to n do
    net ← AñadirNodo(net);
    for j ← 1 to m do
        k ← Seleccionar(δ); // Muestreo sin reemplazo
            proporcional a δ
        δ[k] ← δ[k] + 1;
        net ← AñadirEnlace(net, i, k);
    end
    δ[i] ← m;
end
return net
```

análisis de supervivencia para modelar esperanzas de vida o tiempos de fallo en dispositivos mecánicos [222] y proporciona una generalización bastante buena de la distribución exponencial: la tasa de riesgo (la frecuencia con la cual tienen lugar los fallos) ya no necesita ser constante; realmente, estas tasas se pueden incrementar o decrementar con el tiempo de acuerdo a una cierto parámetro de forma denominado η. Para $\eta = 1$ se obtiene la distribución mencionada anteriormente (exponencial o sin memoria), mientras que para $\eta > 1$ (resp. $\eta < 1$) la tasa de riesgo se incrementa (resp. decrece) con el tiempo –ver Figura 4.4 (a). En los experimentos se ha considerado $\eta > 1$, lo que implica que cuando más tiempo lleva disponible un nodo, más factible es que falle en un futuro cercano. En términos matemáticos, la probabilidad de que un nodo esté disponible en un instante de tiempo dado t_1 suponiendo que ese mismo nodo lo estuvo en el instante de tiempo t_0 es:

$$p(t_0, t_1, \eta, \beta) = e^{-[(t_1/\beta)^\eta - (t_0/\beta)^\eta]} \tag{4.1}$$

donde η y β son los parámetros de forma y escala de la distribución respectivamente. Según este modelo, la duración de la disponibilidad media viene dada por $\beta\Gamma(1+1/\eta)$, donde $\Gamma(\cdot)$ es la función gamma. Por simplicidad, se asume este mismo modelo para la recuperación de los nodos. Considérese que esto vale como ejemplo en el caso de sistemas de computación voluntaria, en los cuales los nodos de computación contribuyen a la computación total cuando están inactivos, ya que como es bien conocido la duración de las tareas de un humano en un compu-

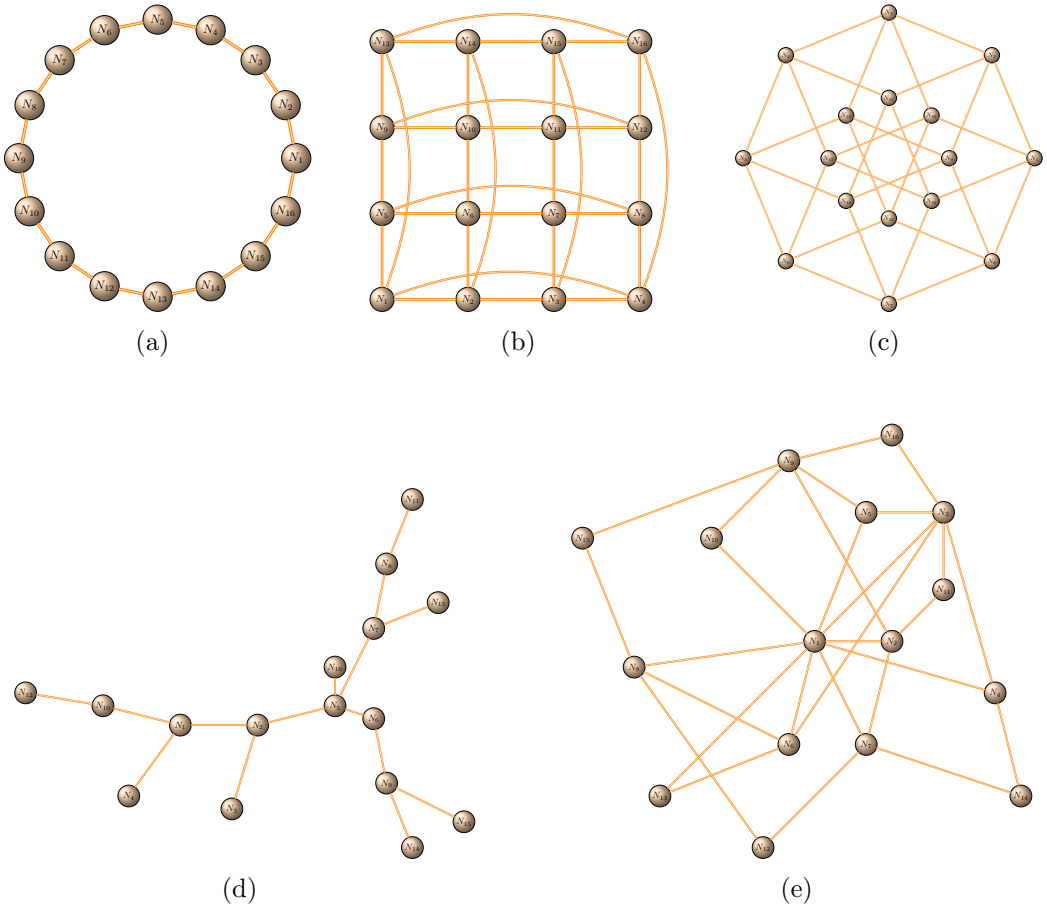
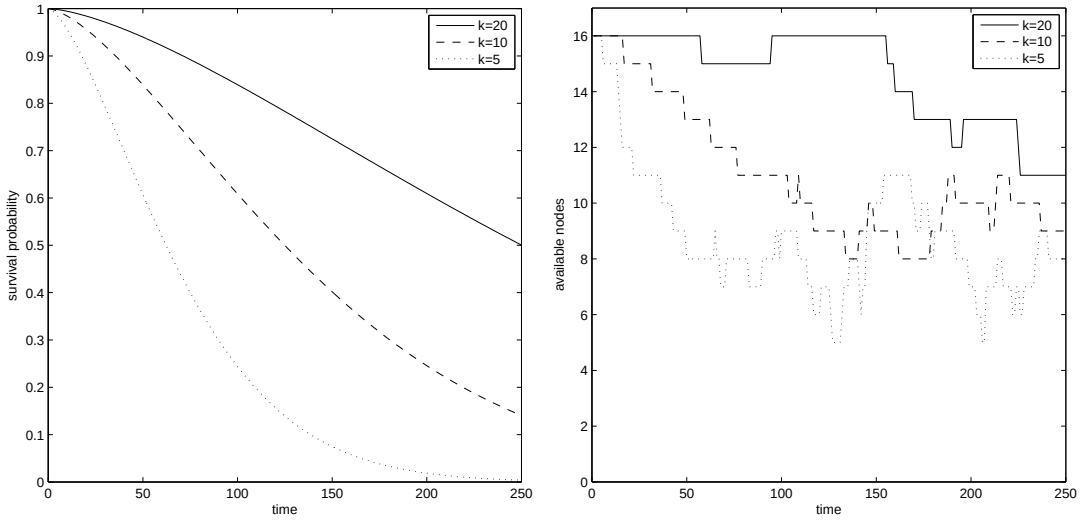


Figura 4.3: Topologías consideradas para la interconexión de las islas. (a) Anillo. (b) Von Neumann. (c) Hipercubo. (d) Escala libre ($m = 1$). (e) Escala libre ($m = 2$).

tador sigue una distribución de Weibull, como se puede ver en [225, 340]. Un ejemplo de cómo el número de nodos disponibles fluctúa usando este modelo se muestra en la Figura 4.4 (b).

Cuando se produce un fallo el nodo correspondiente abandona el sistema. A diferencia de [172], en este caso no se reconectan las interconexiones entre las islas. Por un lado, mientras la forma de actuar en caso de reconexión es conceptualmente bastante directa en casos como la topología en anillo, no está tan claro cómo debería acometerse tal acción de modo más general, preservando las propiedades de la topología subyacente (es decir, la topología puede degenerar en algo totalmente diferente después de unas pocas reconexiones al reducirse su tamaño, lo que introduciría ruido en el análisis de los experimentos). Por otro lado, mantener la topología estática permite centrarse más en el estudio de la robustez inherente de cada esquema de interconexión y además es una estrategia conceptualmente más simple.



(a) (b)

Figura 4.4: (a) Probabilidades de disponibilidad de los nodos según la distribución de Weibull para las configuraciones utilizadas en los experimentos –ver Sección 4.2.4. (b) Ejemplo de la evolución del número de nodos disponibles en una simulación del sistema.

4.2.3. Estrategias de Gestión de Fallos

Las políticas de gestión de fallos tienen como función determinar en qué condiciones vuelve a estar disponible un nodo después de haber atravesado una fase de inactividad (en algunos casos usando información obtenida durante una fase previa de funcionamiento del propio nodo). Más concretamente, se han considerado las siguientes posibilidades:

- no acción: cualquier nodo que se encuentra inactivo se ignora, incluso si vuelve a entrar en la red. Esta es la opción más sencilla y se puede utilizar como patrón para calibrar el resto de estrategias.

- checkpoint: el funcionamiento de cada nodo se monitoriza y se crea un punto de control periódicamente para almacenar su estado. Cuando un nodo vuelve a estar disponible de nuevo, su actividad se reanuda a partir del último punto de control grabado. Esta es la aproximación más clásica para gestionar los fallos en un sentido más general. En los experimentos realizados se ha considerado que esta estrategia crea un punto de control en cada generación del EA.

- reinicialización: las islas que se localizan en nodos que se encuentran disponibles de nuevo se reinicializan de alguna manera. Se han considerado las dos estrategias de reinicialización siguientes:

- aleatoria: se crea la isla a partir de la configuración inicial, al igual que se hace al comienzo de la ejecución del algoritmo.
- probabilística: cuando un nodo se ha reactivado, sus vecinos activos –en función de la topología utilizada– se utilizan para reconstruir la nueva isla (si no hay poblaciones activas adyacentes se ejecuta una re-inicialización aleatoria). Más concretamente, se construye un modelo probabilístico usando las poblaciones adyacentes y a continuación se muestrean para crear la nueva población, siguiendo el espíritu de los EDAs [186, 218, 299] –ver Sección 3.4. En la experimentación, se ha considerado el uso de COMIT [31] para este propósito. Se trata de un EDA que crea una distribución de probabilidad usando dependencias de dos variables y cuya utilidad ya fue probada en la Sección 3.4 obteniendo buenos resultados en el contexto de la optimización multimemética. La lógica de esta estrategia es la creación de un estado de la población que sitúa a la isla en una nivel de demes análogo al de las islas vecinas en la etapa de búsqueda actual, por lo que se evita la reinicialización aleatoria inicial.

La principal ventaja de estas estrategias de reinicialización es el hecho de que no requieren una monitorización activa para crear el punto de control. Por lo tanto, se pueden implementar de forma mas sencilla y además introducen menos sobrecostes computacionales.

Una vez presentados los algoritmos y las estrategias de gestión de fallos, en la próxima sección se describe la experimentación realizada con el objetivo de evaluar estas estrategias.

4.2.4. Análisis Experimental

Con el objetivo de testear la robustez de las diferentes topologías y el rendimiento de las estrategias de gestión de fallos, se ha utilizado un juego de problemas compuesto por cuatro problemas diferentes definidos sobre cadenas binarias, concretamente la función trampa de Deb (TRAP), el problema MMDP y las funciones H-IFF y H-XOR –ver Apéndice A. Los experimentos se han realizado con GAs y MMAs basados en islas compuestos de $n_\iota = 16$ islas interconectadas unas con otras mediante las topologías consideradas en las secciones precedentes. En el caso de la topología SF se ha considerado como valores del parámetro m, $m = 1$ (SF_1) y $m = 2$ (SF_2) para generar la red (se genera una nueva red en cada ejecución del algoritmo). Con respecto a la topología VN, se ha considerado una rejilla toroidal de tamaño 4×4. Cada isla tiene una población de tamaño $\mu = 16$ individuos, siguiendo una plan reproductivo generacional con torneo binario para la selección de los padres, operador de cruce de un punto ($p_X = 1{,}0$), mutación bit-flip ($p_M = 1/\ell$, donde $\ell = 128$ es el numero de islas), búsqueda local cuando sea requerida (guiada usando el meme enlazado al individuo) y reemplazo del

peor padre [269]. Los descendientes heredan el meme del mejor padre, el cual se somete a continuación a un proceso de adaptación de su longitud con una probabilidad p_r y una mutación con probabilidad p_M. Se han considerado memes con longitudes entre $l_{\text{mín}} = 3$ y $l_{\text{máx}} = 9$, con $p_r = 1/l_{\text{máx}}$ para la auto-adaptación de la longitud. La migración (implementada como se indica en la Sección 4.2.2) se ha realizado cada 20 generaciones, permitiendo, por lo tanto, un intervalo de tiempo razonable de evolución de las islas aisladas para cada deme. Una ejecución termina cuando se alcanzan las 50000 evaluaciones.

Con respecto a la simulación de fallos, se ha utilizado el modelo descrito en la sección 4.2.2, usando $\eta = 1{,}5$ como parámetro de forma para generar una tasa de fallo creciente. El parámetro de escala β se utiliza para modular la frecuencia de los fallos. El valor de este parámetro se ha elegido para tener de media un nodo fallando/recuperándose cada k generaciones bajo una hipotética distribución de fallos independiente del tiempo (la distribución actual no es, obviamente, independiente del tiempo ya que $\eta > 1$, pero esta analogía proporciona un punto de referencia para interpretar el valor del parámetro, aunque sea solo de un modo aproximado). Dado que hay n_ι nodos, esto implica que la probabilidad de supervivencia por generación en este escenario hipotético debería ser $p = 1 - (kn_\iota)^{-1}$, que trasladado al parámetro de escala sería $\beta = -1/\log(p)$ (una buena aproximación es $\beta \simeq kn_\iota - 1/2$). Se han considerado tres escenarios diferentes en cuanto al incremento de la volatilidad: $k = 20$ (baja volatilidad), $k = 10$ (volatilidad moderada) y $k = 5$ (alta volatilidad) –ver Figura 4.4 (b). El caso $k = \infty$ representaría la ejecución de los algoritmos sin fallos. Para medir el rendimiento de cada ejecución se ha tomado el mejor individuo del último conjunto de poblaciones. Los experimentos se han realizado con 20 ejecuciones por problema, algoritmo, configuración (topología de las islas y estrategia de gestión de fallos) y escenario de fallos, lo que ha supuesto un total de 10400 ejecuciones.

En las Tablas B.1-B.8 del Apéndice B.1 se proporciona un conjunto completo de datos numéricos con todos los experimentos. No obstante, en las Figuras 4.5-4.8 se proporciona un resumen de todos estos datos. Además de lo anterior, el fitness se ha normalizado usando el fitness medio de $k = \infty$ (sin fallos) como referencia, con el objetivo de conseguir figuras con degradación relativa. Esto permite agregar los resultados de los cuatro problemas del juego de test para obtener las curvas de degradación específicas de cada configuración ⟨topología, estrategia⟩ (Figuras 4.5 y 4.6).

Como se esperaba, la pendiente máxima de degradación la tenemos en no acción –Figuras 4.5 (a) y 4.6 (a). Por el contrario, checkpoint ofrece la degradación mas amigable para todas las topologías –Figuras 4.5 (b) y 4.6 (b)– con una pérdida en el escenario mas drástico de alrededor del 5 % para el GA y solo del 2 % para el MMA. Las dos estrategias de reinicialización ofrecen resultados ligeramente peores (la importancia estadística de esta diferencia se analizará con posterioridad), pero lo más interesante es que parecen ser más sensibles a la topología de interconexión, en particular en el caso del MMA –ver Figuras 4.6 (c)-(d). Especialmente en el caso de reinicialización probabilística, se puede ver como el

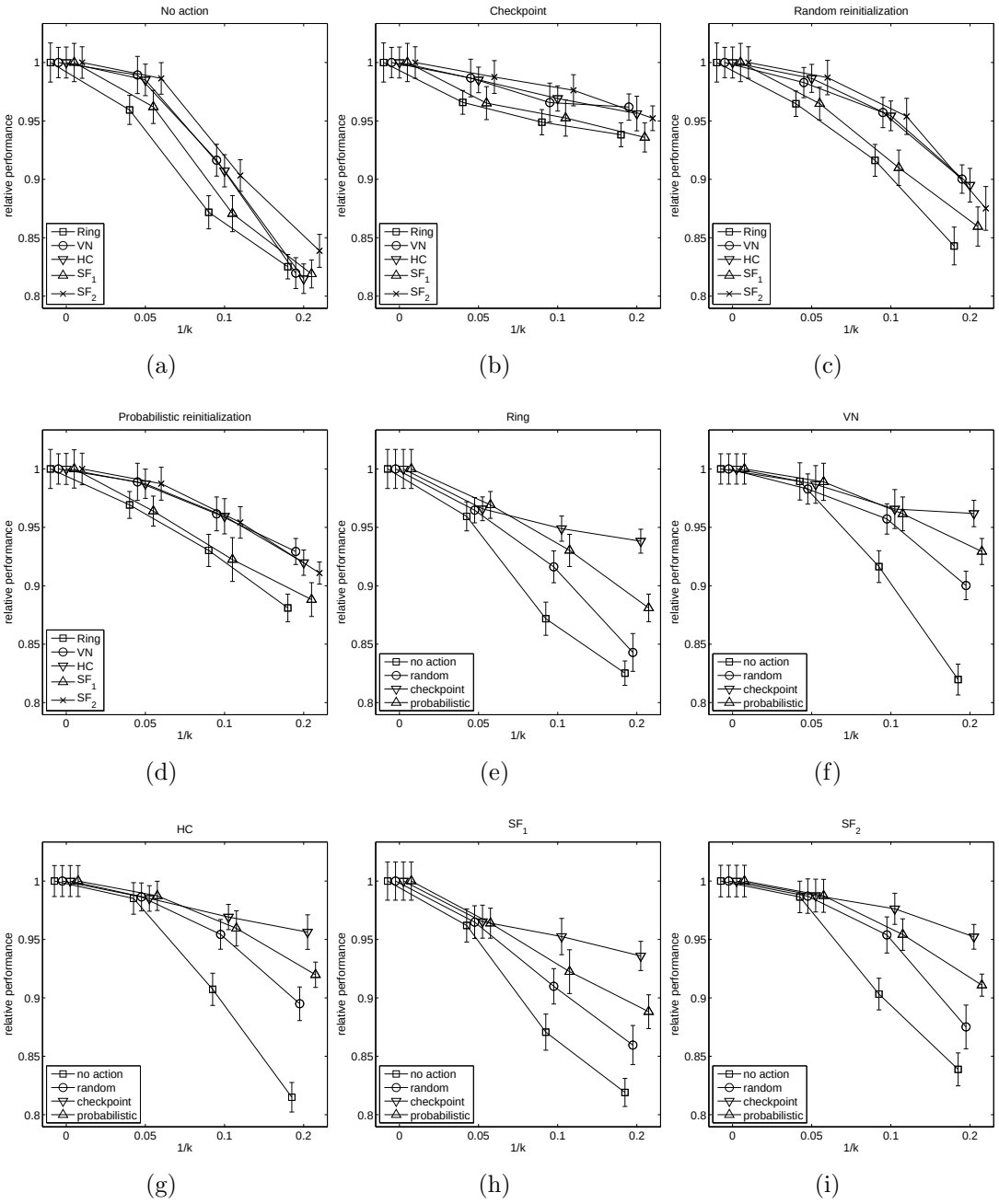


Figura 4.5: Degradación del fitness en GAs basados en islas en función de políticas de gestión de fallos específicas (a)-(d) y topologías (e)-(i).

rendimiento es totalmente competitivo con respecto a checkpoint cuando se usan las topologías VN o HC. Este efecto se puede atribuir a su elevada conectividad, que permite un modelado probabilístico de las islas vecinas más efectivo cuando un nodo se reactiva.

Si se analizan los datos desde la perspectiva complementaria de las topologías,

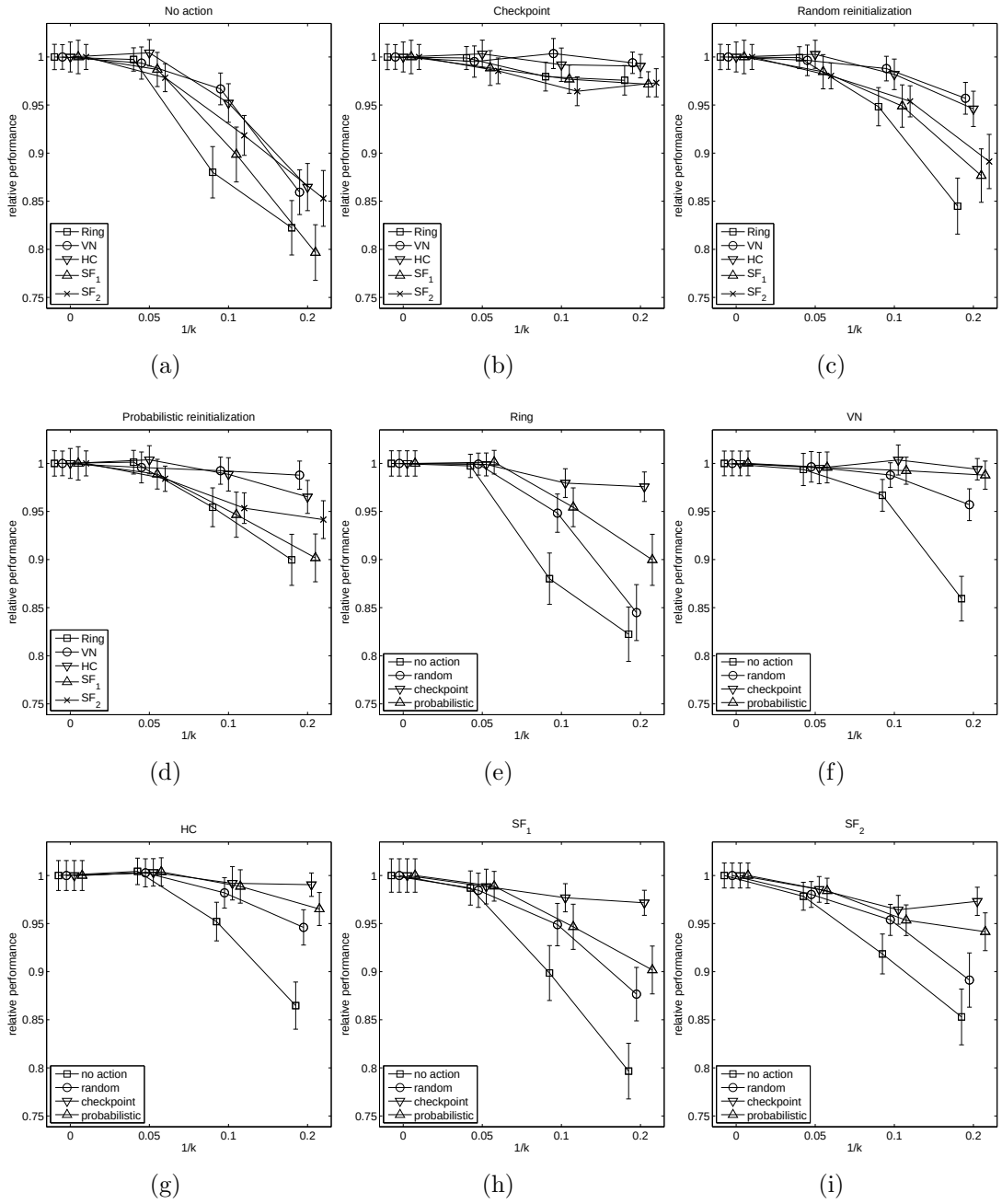
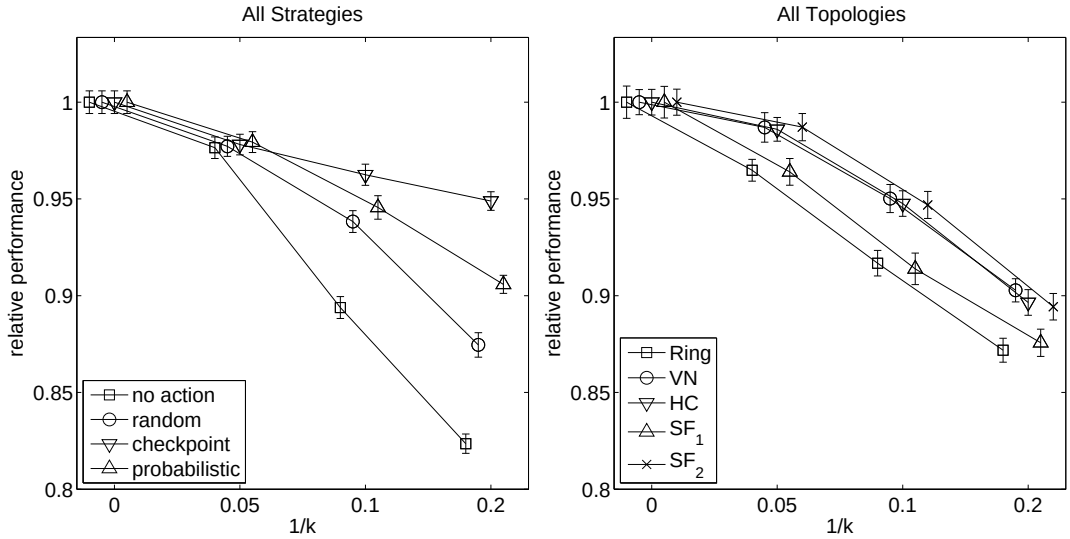


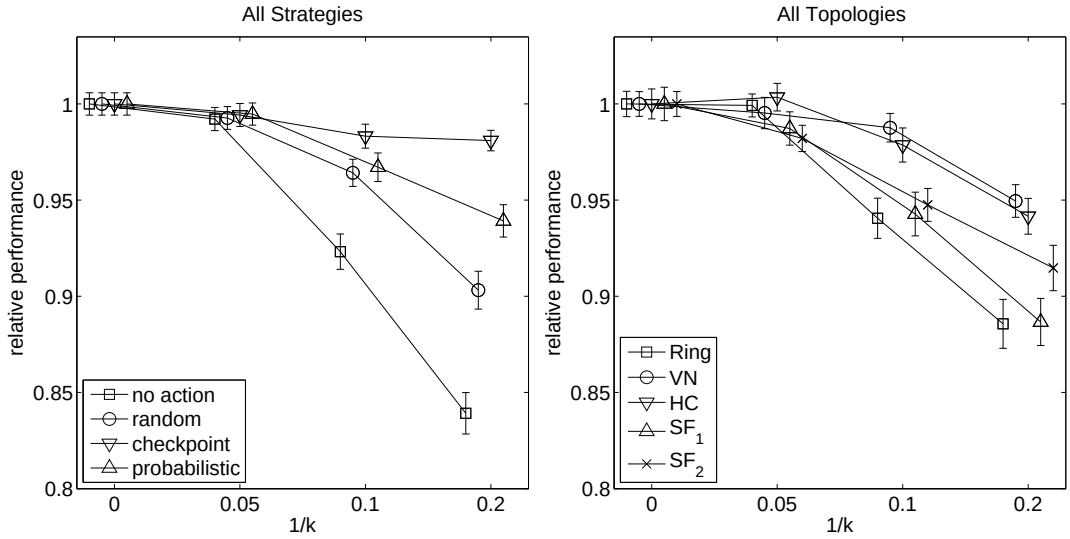
Figura 4.6: Degradación del fitness en MMAs basados en islas en función de políticas de gestión de fallos específicas (a)-(d) y topologías (e)-(i).

se puede observar que en general las características de la degración van a ser más dependientes de la estrategia de gestión de fallos empleada que de cualquier otra variable. Esto es válido tanto para el GA –Figuras 4.5 (e)-(i)– como para el MMA –Figuras 4.6 (e)-(i)– aunque hay que considerar que en el último caso la diferencia entre checkpoint y las dos estrategias de reinicialización es menos



(a) (b)

Figura 4.7: Compendio de la degradación del fitness en GAs basados en islas. (a) En función de la política de gestión de fallos. (b) En función de la topología.



(a) (b)

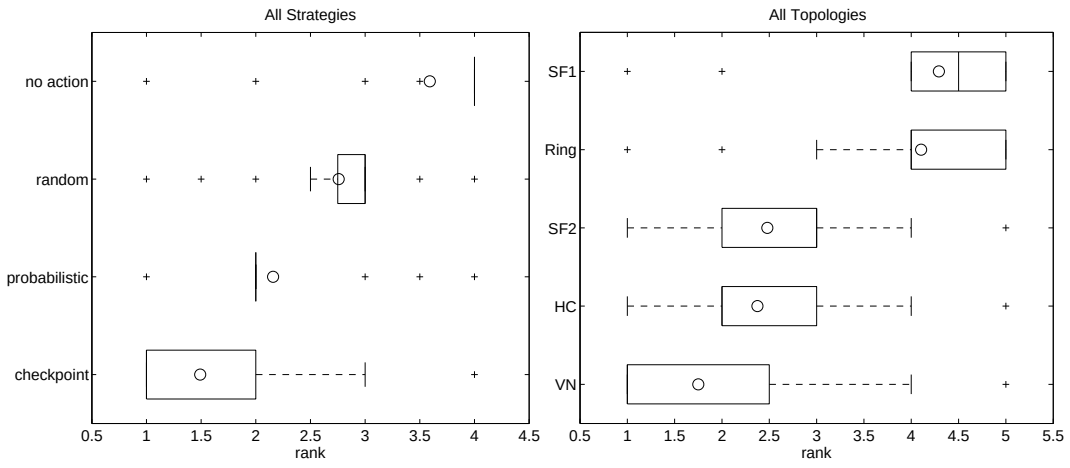
Figura 4.8: Compendio de la degradación del fitness en MMAs basados en islas. (a) En función de la política de gestión de fallos. (b) En función de la topología.

pronunciada que para los GAs con idéntica topología. Esto puede ser debido a que las capacidades de búsqueda mejoradas de un MMA permiten a los demes reactivados reencontrarse más fácilmente con sus vecinos. Además, mientras VN, HC y SF_2 parece que proporcionan un rendimiento robusto solo cuando están en

combinación con checkpoint en el caso del GA –Figuras 4.5 (f)(g) y (i), sí son capaces de proporcionar resultados competitivos cuando se trata de un MMA en combinación con reinicialización probabilística –Figuras 4.6 (f)(g) y (i). Se puede ver también que hay un par de casos en los que el aspecto de la degradación es ligeramente no monótono. Esto es debido a la naturaleza estocástica de estas técnicas y a la perturbación introducida por la volatilidad del entorno y la política de gestión de fallos empleada (como se apuntó en [321], ya que los fallos pueden también jugar un papel importante en el incremento de diversidad durante el proceso de búsqueda en el GA, lo que puede tener consecuencias beneficiosas). No obstante, en cualquier caso estas fluctuaciones no monótonas no son significativas estadísticamente (ver Tablas B.1-B.8). Desde una perspectiva general, las Figuras 4.7 y 4.8 proporcionan una imagen global del comportamiento de ambos modelos algorítmicos. Como se avanzó con anterioridad, el MMA proporciona un rendimiento mejor y una menor degradación que el GA para una misma topología o política de gestión de fallos. Tampoco sorprende que las topologías de anillo y SF_1 sean las que tienen una mayor sensibilidad a la inestabilidad de la red, ya que son las más propensas a ser particionadas en clusters desconectados como resultado de los nodos que fallan. Una vez dicho esto, simplemente aumentar la conectividad no hace que se diluyan las diferencias, ya que hay importantes diferencias entre las restantes topologías. A continuación se aborda esta cuestión.

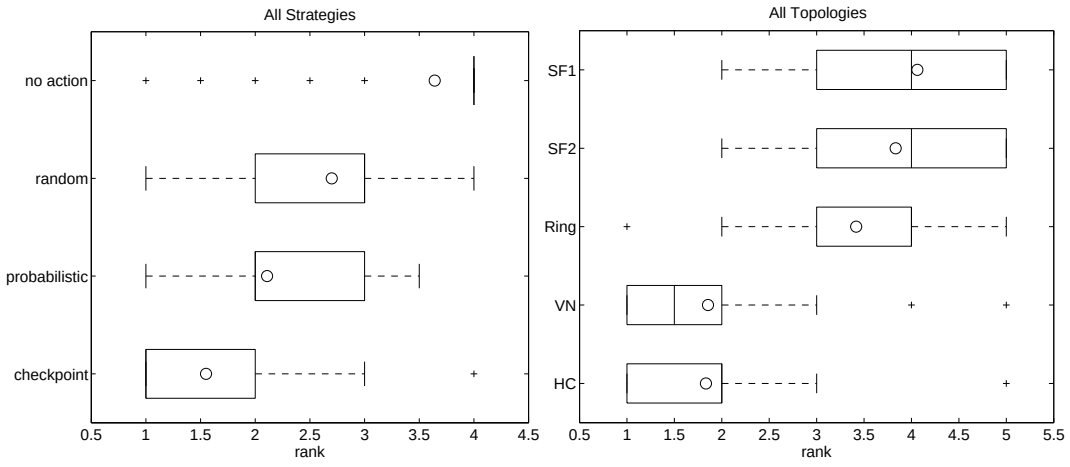
Una vez analizadas las tendencias de la degradación, hay que centrarse en la importancia estadística de los datos y en cómo se deben comparar unos con otros. Para hacer esto se ha realizado un análisis de rankings, calculando el orden relativo de las diferentes estrategias para cada problema, escenario de fallo y topología (y a la inversa, el orden relativo de las diferentes topologías para cada problema, escenario de fallo y estrategia). Se ha asignado ranking 1 a la estrategia (topología) con el mejor rendimiento medio para cada configuración y ranking 4 (ranking 5 en el caso de topologías) para el peor (en caso de empate se tiene en cuenta el ranking medio de las posiciones que conforman el empate). La distribución de los rankings se muestra en las Figuras 4.9-4.12. Una vez hecho esto, se ha utilizado el test de Quade –se trata de un test no paramétrico más sensible que el test de Friedman, ver en [92]– para determinar si hay diferencias significativas entre estos rankings. La Tabla 4.5 muestra el resultado del test para cada una de las comparaciones: estrategia para una topología dada, topología para una estrategia dada, o global (estrategia para todas las topologías y viceversa). Como se puede ver, los p-valores son muy pequeños en todos los casos, sustentando la importancia de las diferencias. Adicionalmente, se ha realizado un test post-hoc para avalar estos resultados, concretamente se ha utilizado el test de Holm para este propósito y el resultado se muestra en la Tabla 4.6.

Inicialmente se considerará una perspectiva global. Para ambos algoritmos, tanto para el GA como para el MMA, checkpoint es la estrategia con mejor ranking, lo cual es estadísticamente significativo ($\alpha = 0{,}05$), como se puede ver en la Tabla 4.6 (primera columna, mitad superior). Con respecto a la topología, no hay un claro vencedor ya que el test no lo pasan las topologías de HC y SF_2 con



(a) (b)

Figura 4.9: Distribución global de rankings de los índices de degradación del rendimiento en GAs basados en islas (a) En función de la política de gestión de fallos. (b) En función de la topología.



(a) (b)

Figura 4.10: Distribución global de rankings de los índices de degradación del rendimiento en MMAs basados en islas (a) En función de la política de gestión de fallos. (b) En función de la topología.

respecto a VN para el GA, ni VN con respecto a HC en el caso del MMA. Esto indicaría, en primer lugar, que checkpoint es claramente la mejor estrategia y que VN/HC son las topologías más robustas. Por otro lado, checkpoint proporciona un alto rendimiento para todas las topologías, así como HC y VN para todas las estrategias de gestión de fallos. Sin embargo, si se mira con más atención la intersección de los dos ganadores, la situación admite algunos matices: para ambos, tanto para el GA como para el MMA con topologías VN y HC, checkpoint no lo-

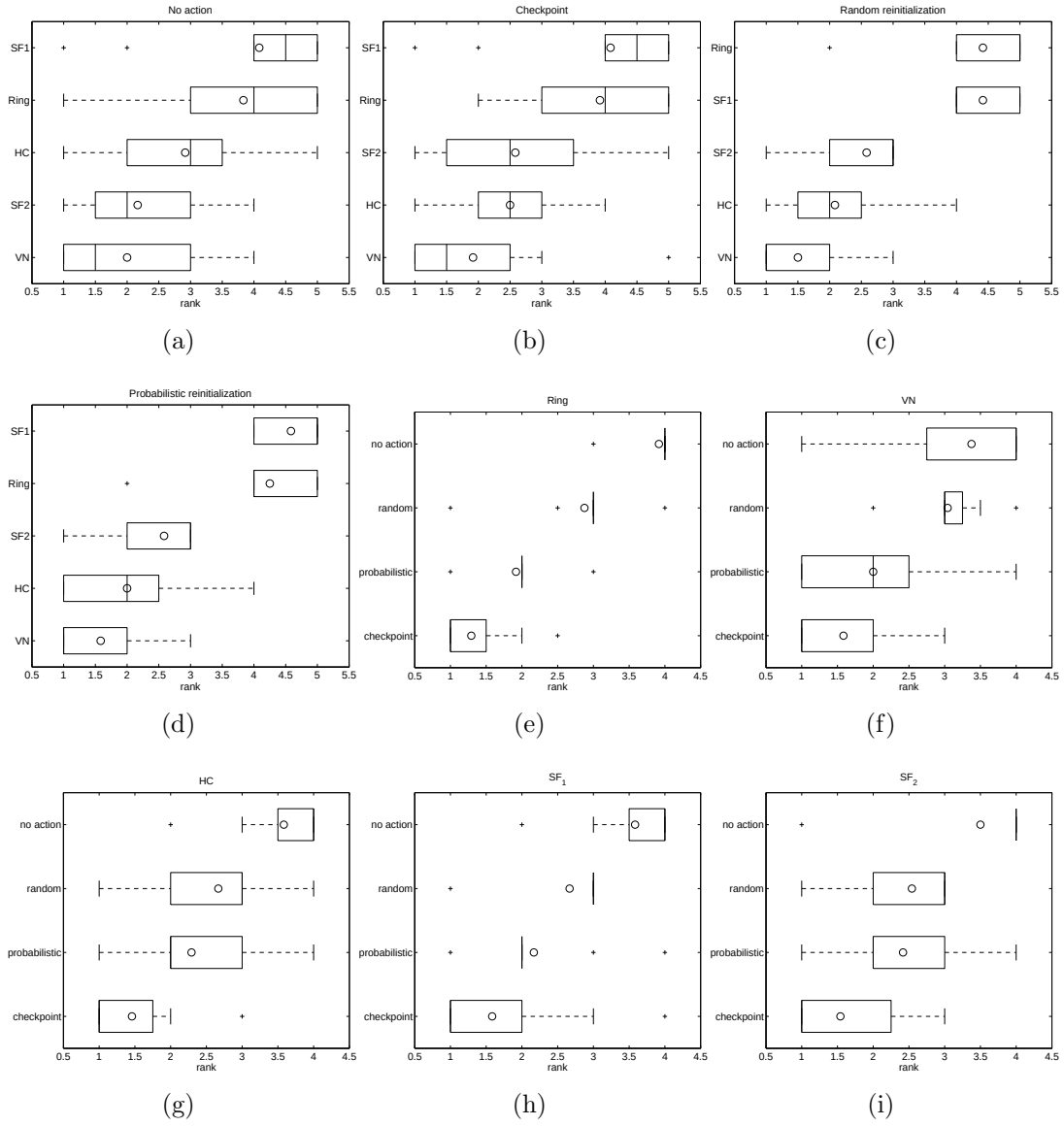


Figura 4.11: Distribución global de rankings de los índices de degradación del rendimiento en GAs basados en islas por políticas de gestión de fallos concretas (a)-(d) y por topologías (e)-(i).

gra diferencias estadísticamente significativas con reinicialización probabilística (y de hecho, esta ultima se clasifica en primer lugar en el MMA con topología VN). Considérese también que tanto HC como VN no tienen diferencias significativas cuando se utiliza como estrategia de gestión de fallos tanto checkpoint como reinicialización probabilística. En general, esto sugiere que checkpoint es realmente una estrategia poderosa, pero su superioridad global viene marcada por su robustez cuando trata con topologías frágiles o de rendimiento pobre. Este es verdaderamente uno de los puntos fuertes de esta estrategia, pero también es interesante

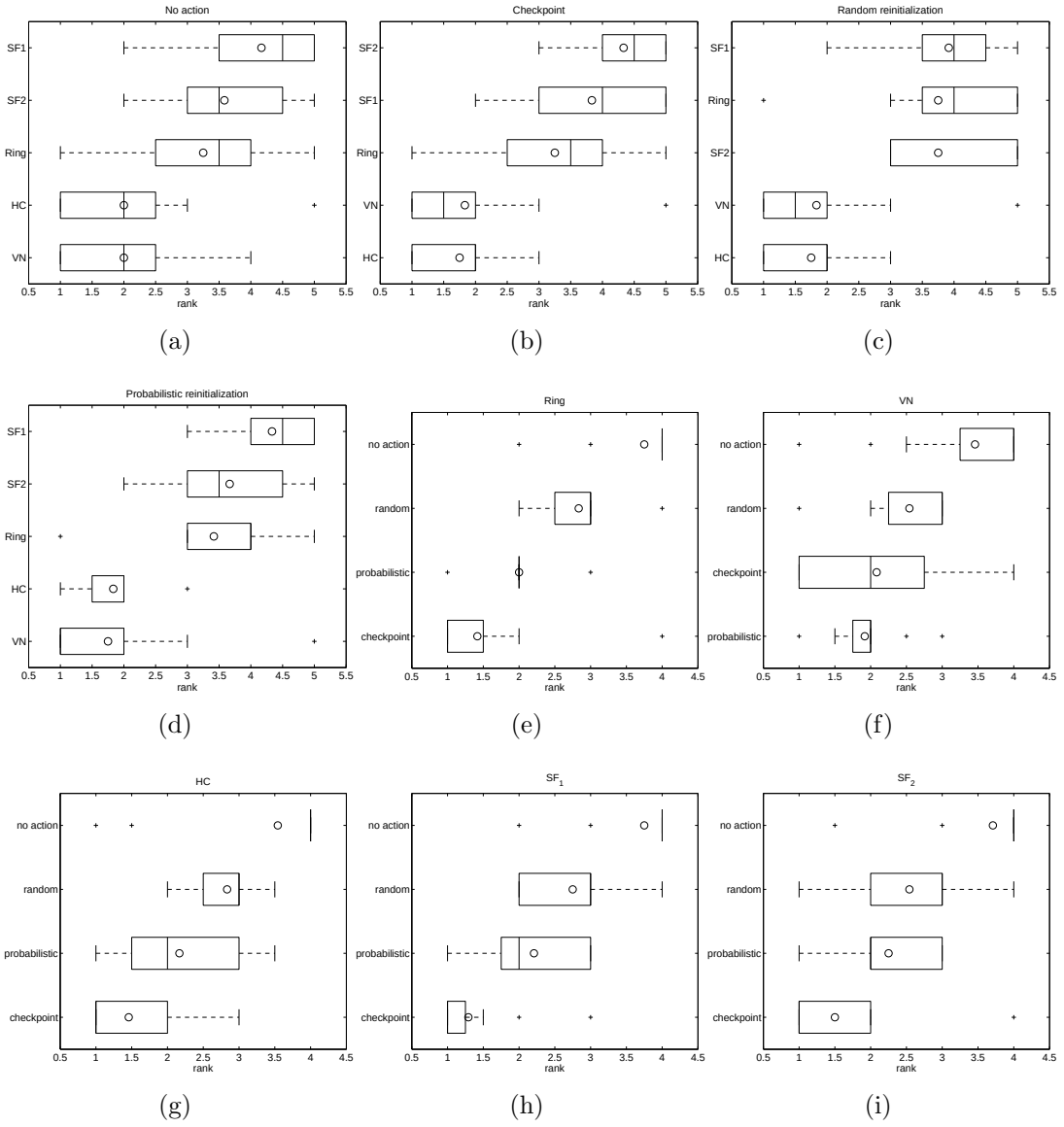


Figura 4.12: Distribución global de rankings de los índices de degradación del rendimiento en MMAs basados en islas por políticas de gestión de fallos concretas (a)-(d) y por topologías (e)-(i).

reseñar que cuando la topología se selecciona adecuadamente (se elige una de mayor rendimiento), la estrategia reinicialización probabilística ofrece resultados comparables. Esto es interesante si tenemos en cuenta que esta última estrategia puede ser bastante más simple de implementar en la práctica, ya que no requiere grabar periódicamente el estado de las islas en algún almacén de datos estable.

Tabla 4.5: Resultados (p-valores) del test estadístico de Quade en función de los datos de degradación del rendimiento de los GAs/MAs basados en islas. La mitad superior (resp. mitad inferior) corresponde a la comparación entre las diferentes políticas de gestión de fallos (resp. topologías) para una topología determinada (resp. estrategia) y para todas ellas.

		Topología				
	Todas	Anillo	VN	HC	SF_1	SF_2
GA	0.00e-00	1.58e-08	1.89e-05	6.42e-06	7.81e-07	3.75e-06
MMA	0.00e-00	2.73e-06	1.46e-04	2.65e-05	9.16e-07	6.66e-07

		Política de gestión de fallos			
	Todas	no acción	checkpoint	aleatoria	probabilística
GA	0.00e-00	2.29e-04	2.19e-04	2.45e-08	5.11e-08
MMA	0.00e-00	8.65e-05	2.13e-04	4.44e-06	1.17e-06

4.2.5. Discusión

La tolerancia a fallos y la resistencia al cambio son indudablemente dos cuestiones de suma importancia en los EAs basados en islas dado que incrementan la relevancia de los entornos emergentes para computación paralela, tales como las redes P2P o los entornos tipo *grid*. En este sentido, considerando el comportamiento de dos variantes de EAs en presencia de recursos volátiles y teniendo en cuenta diversos factores de diseño como la topología de interconexión o la estrategia de gestión de fallos empleada, los resultados indican en primer lugar que este modelo de granularidad gruesa es aparentemente robusto (se han podido confirmar los resultados ya anticipados en la literatura desde una perspectiva diferente, por ejemplo en [172]), tolerando escenarios tanto de baja como moderada volatilidad y permitiendo más la incorporación de estrategias de gestión de fallos para los escenarios inestables. Relacionado con esto, las topologías estructuradas regularmente como las rejillas de von Neumann o los hipercubos parecen proporcionar un perfil más gradual de degradación. Desde el punto de vista de la política de gestión de fallos, la estrategia denominada checkpointing parece ser la elección más robusta, ya que es capaz de proporcionar un buen rendimiento en la mayoría de los escenarios. Sin embargo, se ha mostrado también muy competitiva la estrategia de reinicialización probabilística basada en chequear los vecinos activos cuando un nodo se reactiva y usar esta información para construir la nueva isla, proporcionando resultados comparables a los obtenidos por la estrategia de checkpoint, sin requerir crear puntos de control periódicos que habría que ir grabando en algún dispositivo externo. Por otro lado, el análisis estadístico de los resultados sugiere una tolerancia a fallos inherente a estos EAs e indica que, a pesar de que la estrategia checkpoint es la más robusta y es superior cuando se utilizan las topologías más frágiles, una estrategia de reinicialización más simple

Tabla 4.6: Resultados del test de Holm ($\alpha = 0{,}05$). En cada entrada de la tabla se indica en primer lugar y en negrita la topología/estrategia de control. A continuación se enumeran las estrategias/topologías restantes en orden según su ranking, usando letra cursiva para denotar aquellas que no pasan el test con respecto a la que actúa como control (la que tiene el mejor ranking) y en letra estándar se muestran aquellas que han pasado el test (una línea horizontal separa las primeras de las últimas). La mitad superior (resp. mitad inferior) corresponde a las diferentes políticas de gestión de fallos (resp. topologías) para una topología determinada (resp. estrategia) y para todas ellas.

		Topología				
	Todas	anillo	VN	HC	SF_1	SF_2
GA	**checkpoint**	**checkpoint**	**checkpoint**	**checkpoint**	**checkpoint**	**checkpoint**
	probabilística	*probabilística*	*probabilística*	*probabilística*	*probabilística*	*probabilística*
	aleatoria	aleatoria	aleatoria	aleatoria	aleatoria	*random*
	no acción	no acción	no acción	no acción	no acción	no acción
MMA	**checkpoint**	**checkpoint**	**probabilística**	**checkpoint**	**checkpoint**	**checkpoint**
	probabilística	*probabilística*	*checkpoint*	*probabilística*	probabilística	*probabilística*
	aleatoria	aleatoria	*aleatoria*	aleatoria	aleatoria	aleatoria
	no acción	no acción	no acción	no acción	no acción	no acción

		Política de gestión de fallos			
	All	no acción	checkpoint	aleatoria	probabilística
GA	**VN**	**VN**	**VN**	**VN**	**VN**
	HC	*SF_2*	*HC*	*HC*	*HC*
	SF_2	*HC*	*SF_2*	*SF_2*	*SF_2*
	SF_1	anillo	anillo	SF_1	anillo
	anillo	SF_1	SF_1	anillo	SF_1
MMA	**HC**	**VN**	**HC**	**HC**	**VN**
	VN	*HC*	*VN*	*VN*	*HC*
	anillo	*anillo*	anillo	SF_2	anillo
	SF_2	SF_2	SF_1	anillo	SF_2
	SF_1	SF_1	SF_2	SF_1	SF_1

aparentemente proporciona resultados comparables estadísticamente a la anterior cuando se utilizan topologías de mayor rendimiento, como por ejemplo las basadas en rejilla de von Neumann o hipercubos.

4.3. Análisis de Sensibilidad de la Estrategia de Checkpoint

Como se vio en la sección anterior, una estrategia basada en la grabación del estado actual de la población en almacenamiento externo cada cierto tiempo puede ser una opción satisfactoria como medida preventiva ante la caída de una parte de los nodos. En las siguientes secciones se analiza con detalle esta aproxi-

mación sobre diversos tipos de redes de escala libre y de mundo pequeño[2] y bajo diferentes escenarios de volatilidad.

4.3.1. Topologías de Red

Como en los experimentos anteriores se va a utilizar un MMA como el definido en la Sección 3.1, con las islas distribuidas sobre una red de nodos y con el proceso de migración ejecutado de forma asíncrona (eligiendo aleatoriamente un individuo de una isla y transfiriéndolo a otra donde reemplaza al peor individuo de la isla destino). El escenario computacional tiene dos factores fundamentales: la topología de interconexión y el modelo dinámico de la red. Con respecto a la topología, se consideran dos posibilidades:

- Redes de escala libre (SF): se caracterizan por la existencia de una distribución que sigue una ley potencial en el grado de los nodos –ver Sección 4.2.2 para más detalles.

- Redes de mundo pequeño (SW): se caracterizan porque las distancias medias entre los nodos son muy pequeñas (a menudo $\mathcal{O}(\log n_\iota)$ donde n_ι es el número de nodos) y tienen un coeficiente de agrupamiento[3] elevado (los vecinos de un nodo es muy probable que sean vecinos de ellos mismos). Existen varios procedimientos para crear este tipo de redes. La aproximación más clásica es el modelo de Watts-Strogatz [358] en el cual se construye un anillo regular y posteriormente se van añadiendo ejes con una probabilidad determinada. En esta tesis doctoral se ha optado por una aproximación diferente para crear estas redes, concretamente por una variante denominada modelo de Barmpoutis-Murray (BM) [36]. Este modelo produce redes ultra-SW con una distancia media muy pequeña y un coeficiente de agrupamiento muy grande. La construcción de la red se controla por un parámetro M que representa el número total de enlaces en la red. El algoritmo intenta crear el clique máximo que deje disponibles los enlaces suficientes para que se puedan seguir conectando el resto de nodos de la red. Este procedimiento se repite de forma recursiva para el resto de nodos y enlaces (si en un paso determinado se encuentra que no se puede construir una subred con el número deseado de enlaces y nodos, el algoritmo da marcha atrás y reduce en una unidad el tamaño del clique del paso anterior –ver Algoritmo 6 para esta parte del procedimiento). En el modelo original de BM todas estas subredes se enlazan a través de un nodo fijo del primer clique, lo que lo convierte en un punto de articulación crítico que reduce la resistencia de la red. Por este motivo se ha optado por modificar el modelo original para tener cada subred enlazada al primer clique mediante un nodo seleccionado aleatoriamente. Para poder establecer comparaciones con las redes SF, las

[2] *small world*
[3] *clustering coefficient*

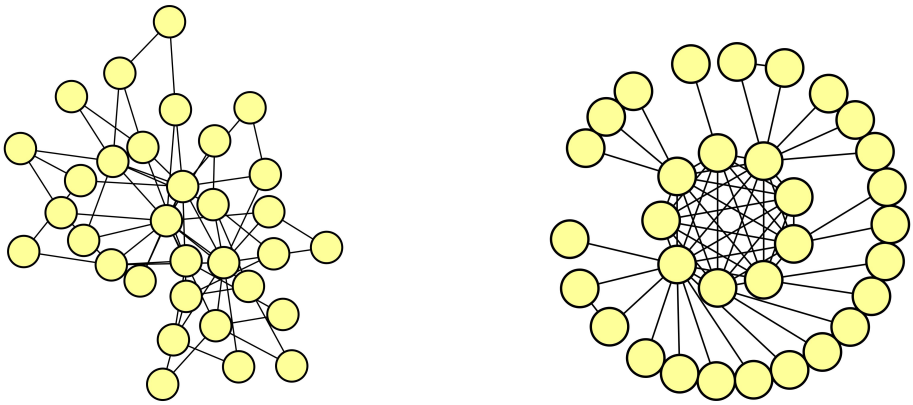


Figura 4.13: Ejemplo de redes con $n_\iota = 32$. La figura de la izquierda es una red de tipo SF ($m = 2$) y la de la derecha es una red SW ($M = 61$).

cuales se ejecutan siguiendo el modelo BA con el parámetro m que genera redes con $M = nm - m(m+1)/2$ enlaces, se ha utilizado este valor M para construir las redes SW análogas.

La Figura 4.13 muestra un ejemplo de ambos tipos de redes con el mismo número de nodos y enlaces.

Con respecto a la dinámica de la red, esta se caracteriza por patrones de disponibilidad de los nodos de cómputo como se vio en la Sección 4.2, donde los n_ι nodos están inicialmente disponibles y a partir de ahí su disponibilidad sigue una distribución de Weibull. En la Sección 4.2 se consideraron diversas políticas de gestión de fallos para mitigar el efecto del *churn*. En el contexto actual, en el que el punto de interés se centra en analizar la estrategia de checkpoint, se va a utilizar también la política de reinicialización aleatoria (en la que los nodos que se vuelven a activar se inicializan desde cero, como en la fase inicial del algoritmo) como referencia para medir el rendimiento de la primera.

Es evidente que la reinicialización aleatoria es una estrategia que tiene una ventaja potencial al reintroducir diversidad en el proceso de búsqueda. Por otro lado, checkpoint tiene la ventaja de que no corta el progreso previo de la búsqueda, lo que permite mantener el impulso. El lado más negativo es que esta última estrategia requiere la utilización de algún tipo de almacenamiento externo para grabar los estados del sistema cada cierto tiempo, lo que puede inducir una sobrecarga adicional en la ejecución del algoritmo (especialmente si hay cuestiones de seguridad y de privacidad involucradas [249]). Con respecto a los periodos λ (medido en número de iteraciones) para realizar la grabación de los estados, estos se pueden ajustar para minimizar la pérdida de rendimiento del sistema. El efecto de este parámetro se estudia en profundidad en la siguiente sección.

Algoritmo 6: Modelo de Barmpoutis-Murray

```
function GenerarSubgrafos (↓ n, m : ℕ) returns ({Grafo}, 𝔹)
// n: Número de nodos, m: Número de enlaces de cada nodo
L ← n(n − 1)/2;
if (m = L) then
    // Grafo completo. Clique es el constructor con n nodos
    return ⟨{Clique(n)}, TRUE⟩;
else
    if (m = (n − 1)) then
        // Star(n) crea un nodo central al que
        // se conectan n-1 nodos
        return ⟨{Star(n)}, TRUE⟩;
    else
        L ← (n − 1)(n − 2)/2;
        if (m ⩾ (2 + L)) then
            // Incomplete(n,m) genera un grafo con un
            // clique de tamaño n-1 y le añade otro
            // nodo con las aristas restantes hasta m
            return ⟨{Incomplete(n, m)}, TRUE⟩;
        else
            OK ← FALSE;
            k ← n − 1;
            while (¬OK ∧ (k > 1)) do
                L ← k(k − 1)/2;
                while (L > m ∨ (m − L) < (n − k)) do
                    k ← k − 1;
                    L ← k(k − 1)/2;
                end
                ⟨S′g, OK⟩ ← GenerarSubgrafos(n − k + 1, m − L);
                if OK then
                    Sg ← {Clique(k)} ∪ S′g;
                else
                    k ← k − 1;
                end
            end
            return ⟨Sg, OK⟩;
        end
    end
end
```

4.3.2. Análisis Experimental

Los experimentos se han realizado con un MMA distribuido con $n_\iota = 32$ islas, en el que cada isla tiene una población de tamaño $\mu = 16$ individuos. La longitud de los memes evoluciona entre $l_{\text{mín}} = 3$ y $l_{\text{máx}} = 9$, variando su longitud con probabilidad $p_r = 1/9$. Se ha considerado una probabilidad para el cruce $p_X = 1{,}0$ (cruce de un punto), probabilidad de mutación $p_M = 1/\ell$ (mutación bit-flip), donde ℓ es la longitud del genotipo y una probabilidad de migración $p_{mig} = 1/80$ (i.e., una migración en promedio cada $80 = 5\mu$ evaluaciones, es decir, cada 5 generaciones completas). Los MMAs ejecutados en cada isla son de estado estacionario. Para generar la topología de la red se ha utilizado el parámetro $m = 2$ en el modelo de BA de redes SF y el valor correspondiente $M = nm - m(m+1)/2$ en el modelo de BM de redes SW para mantener el número de enlaces igual en ambos casos. Con respecto a la dinámica de los nodos, se ha utilizado como parámetro de forma $\eta = 1{,}5$ (con lo que la probabilidad de fallos se incrementa con el tiempo) y como parámetro de escala $\beta = -1/\log p(k)$ con $p(k) = 1-(kn_\iota)^{-1}$, $k \in \{1, 2, 5, 10, 20\}$. De esta forma, el periodo de disponibilidad media por nodo es alrededor del $90\,\% \cdot kn_\iota$ iteraciones. Los escenarios estudiados van en el rango de ($k = 20$) para tasas de *churn* bajas hasta ($k = 1$) para tasas de *churn* extremadamente altas. Para analizar la respuesta de la estrategia de checkpoint se consideran los valores de $\lambda \in \{\mu, 10\mu, 100\mu\}$ donde μ es el tamaño de la población de cada isla. Por último, se han considerado cuatro problemas de test: TRAP, H-IFF, H-XOR y MMDP –ver Apéndice A– y se han ejecutado 25 simulaciones con 50000 evaluaciones cada una para cada valor de λ, escenario de *churn*, problema y topología de red.

La Figura 4.14 muestra los resultados obtenidos en términos de desviación con respecto a la solución óptima (promediados para los cuatro problemas) como función de la tasa de *churn*, independientemente para cada valor de λ y para cada topología de red (los datos numéricos detallados se proporcionan en la Tabla 4.7).

En primer lugar, claramente el rendimiento se degrada conforme aumenta la tasa de *churn*. No obstante, se puede observar que las variantes que utilizan la estrategia de reactivación de checkpoint tienen un rendimiento considerablemente mejor que las de reinicialización aleatoria. Esto confirma los resultados previos sobre este tipo de estrategias –ver Sección 4.2– y valida su utilidad sobre diferentes topologías de red. Considérese sin embargo, que hay una importante degradación del rendimiento conforme disminuye la frecuencia del backup (o conforme aumenta el periodo entre grabaciones del estado de los nodos λ). Esta degradación es estadísticamente significativa de acuerdo con el test de Quade (p-valor ≈ 0) tanto globalmente como cuando se analizan de forma independiente las topologías SF y SW. Si se utiliza el test de Holm para realizar un análisis post-hoc, se observa que el uso de checkpoint con parámetro $\lambda = \mu$ es estadísticamente superior al resto de técnicas en el nivel $\alpha = 0{,}05$ –ver Tabla 4.8. Este resultado sugiere que las estrategias menos costosas (en términos de los requisitos menos frecuentes para realizar la copia del estado del sistema) no son capaces de tratar

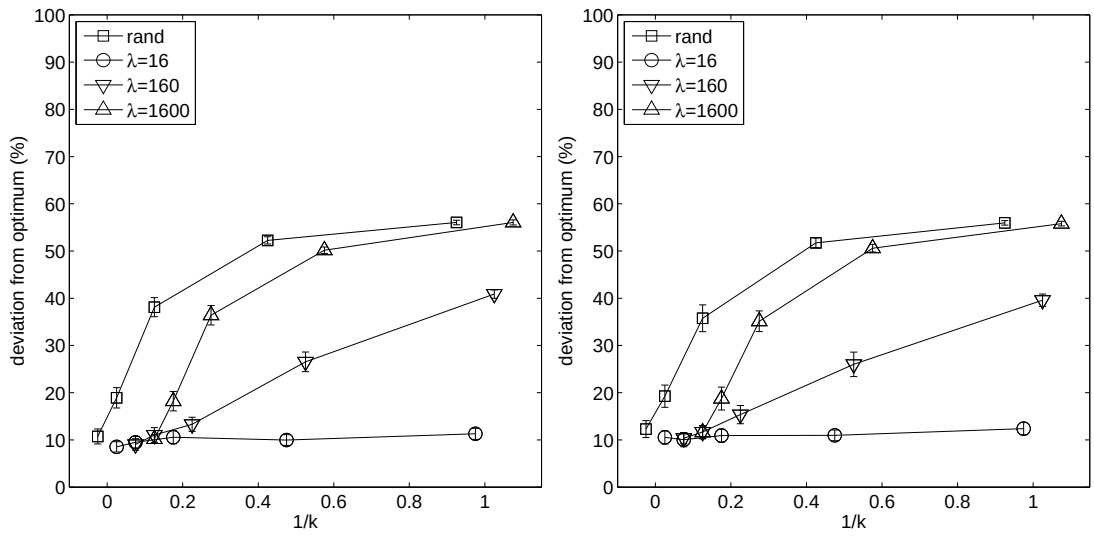


Figura 4.14: Desviación con respecto a la solución óptima como función de la tasa de *churn* para redes (a) SF y (b) SW.

adecuadamente con el *churn* (este resultado también se sostiene si se realiza un análisis separado para cada topología de red estudiada). Las Figuras 4.15 y 4.16 muestran una representación más clara del comportamiento de los MMAs para valores bajos ($k = 20$), altos ($k = 5$) y extremadamente altos ($k = 1$) de la tasa de *churn*. Considérese que en el escenario más estable (Figuras 4.15-c y 4.16-c) el algoritmo todavía funciona de forma robusta independientemente de la frecuencia de la grabación del estado del sistema (aunque como puede verse en la Figura 4.16-c existen considerables diferencias en términos de diversidad genética cuando el periodo λ es mayor). Sin embargo, conforme la tasa de *churn* se incrementa las diferencias en el fitness empiezan a ser remarcadamente elevadas en favor de $\lambda = \mu$. Como se puede ver en la Figuras 4.16-a y 4.16-b, el MMA tiene problemas de convergencia en estos escenarios cuando λ es alto, lo que indica que cuando la frecuencia de las instantáneas es baja el sistema no puede mantener el impulso de la búsqueda en este tipo de entornos inestables (ver Figuras 4.15-a y 4.15-b).

Finalmente se ha realizado un análisis de rankings, calculando el orden relativo de las diferentes estrategias para cada problema y topología. La distribución de los rankings se muestra en la Figura 4.17. El resultado corrobora lo anterior, ya que la última posición es para la estrategia de reinicialización aleatoria y las tres primeras para checkpoint, siendo la primera posición para $\lambda = \mu$, y conforme se incrementa el valor de λ la estrategia de reactivación se va retrasando en el ranking.

Tabla 4.7: Resultados (promediados para 25 ejecuciones) de los diferentes MMAs sobre los cuatro problemas considerados. Se muestra la mediana ($\tilde{x}$), la media ($\bar{x}$) y el error estándar de la media ($\sigma_{\bar{x}}$).

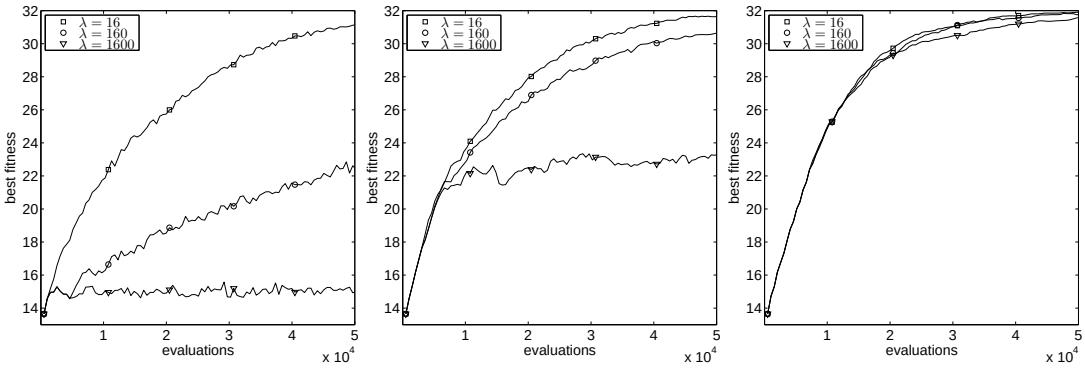
			TRAP		H-IFF		H-XOR		MMDP	
topología	k	λ	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$
SF	20	rand	0.00	1.30 ± 0.36	0.00	5.17 ± 2.22	30.90	31.56 ± 1.03	4.49	4.90 ± 0.49
		1600	1.25	1.32 ± 0.28	0.00	1.56 ± 1.09	34.03	33.93 ± 0.86	3.00	3.88 ± 0.43
		160	0.00	0.75 ± 0.34	0.00	2.81 ± 1.56	30.56	29.10 ± 0.97	3.00	3.52 ± 0.56
		16	0.00	0.30 ± 0.15	0.00	2.02 ± 1.44	29.17	29.06 ± 0.84	3.00	2.74 ± 0.46
	10	rand	9.38	9.35 ± 1.09	0.00	11.84 ± 2.90	40.10	40.47 ± 1.04	14.32	13.96 ± 1.10
		1600	8.75	7.10 ± 0.91	16.67	13.67 ± 2.99	39.41	39.47 ± 0.79	13.15	12.58 ± 0.73
		160	0.00	1.17 ± 0.38	0.00	6.70 ± 2.27	31.77	30.95 ± 0.84	5.66	5.30 ± 0.53
		16	0.00	0.43 ± 0.20	0.00	3.28 ± 1.53	31.94	30.42 ± 0.97	4.16	3.90 ± 0.58
	5	rand	26.25	27.02 ± 1.51	47.40	44.85 ± 2.55	53.30	53.11 ± 0.91	27.79	27.46 ± 0.88
		1600	28.13	27.25 ± 1.43	47.05	42.13 ± 2.62	52.43	50.64 ± 1.14	25.97	25.60 ± 0.79
		160	3.75	4.25 ± 0.79	0.00	4.10 ± 2.00	36.63	35.35 ± 0.81	10.15	9.53 ± 0.63
		16	0.00	1.07 ± 0.35	0.00	5.23 ± 1.97	30.56	30.50 ± 0.93	5.66	5.49 ± 0.52
	2	rand	47.50	47.54 ± 0.75	62.67	61.56 ± 0.68	62.50	62.45 ± 0.33	37.29	37.46 ± 0.62
		1600	42.50	42.86 ± 0.75	60.59	60.58 ± 0.55	61.98	61.83 ± 0.42	35.13	35.37 ± 0.48
		160	17.50	18.10 ± 0.86	24.65	23.59 ± 3.07	46.70	45.82 ± 0.71	19.14	18.60 ± 0.62
		16	1.25	1.40 ± 0.33	0.00	3.22 ± 1.51	30.56	29.90 ± 0.76	5.99	5.38 ± 0.57
	1	rand	53.75	54.08 ± 0.59	64.58	64.52 ± 0.23	64.76	64.43 ± 0.25	41.63	41.14 ± 0.36
		1600	53.75	53.22 ± 0.65	64.76	64.32 ± 0.31	65.10	65.12 ± 0.20	42.29	41.59 ± 0.43
		160	30.00	29.78 ± 0.80	50.35	50.81 ± 0.92	55.21	55.31 ± 0.40	28.30	27.62 ± 0.58
		16	2.50	2.68 ± 0.44	0.00	2.24 ± 1.57	32.47	31.88 ± 0.74	8.99	8.35 ± 0.51
SW	20	rand	2.50	3.79 ± 0.95	0.00	6.56 ± 2.29	34.03	34.17 ± 1.26	4.49	4.65 ± 0.69
		1600	0.00	1.63 ± 0.50	0.00	6.36 ± 1.92	34.72	34.59 ± 0.74	3.00	3.82 ± 0.46
		160	0.00	0.55 ± 0.26	0.00	4.87 ± 1.63	34.72	33.29 ± 0.88	3.00	2.73 ± 0.42
		16	0.00	1.45 ± 0.47	0.00	4.72 ± 1.77	31.25	31.04 ± 1.13	4.49	4.96 ± 0.58
	10	rand	8.13	7.75 ± 1.25	19.44	17.09 ± 3.27	39.76	40.22 ± 0.82	11.65	11.98 ± 1.11
		1600	7.50	8.68 ± 1.18	16.67	15.10 ± 3.36	41.67	40.46 ± 1.15	10.48	10.79 ± 1.05
		160	2.50	3.17 ± 0.65	0.00	4.53 ± 1.69	35.94	34.01 ± 1.15	4.49	4.97 ± 0.52
		16	0.00	1.00 ± 0.32	0.00	5.29 ± 2.06	30.56	29.48 ± 1.01	4.16	4.36 ± 0.49
	5	rand	28.75	29.52 ± 1.47	42.19	34.11 ± 4.09	53.30	53.16 ± 1.06	26.30	26.28 ± 0.80
		1600	18.13	20.75 ± 1.71	47.22	42.76 ± 2.74	50.35	51.20 ± 0.96	26.63	25.80 ± 0.98
		160	2.50	3.54 ± 0.68	0.00	10.78 ± 2.86	37.85	37.54 ± 0.74	10.48	9.59 ± 0.57
		16	0.00	0.92 ± 0.32	0.00	5.22 ± 1.92	34.03	32.76 ± 0.95	4.49	4.73 ± 0.51
	2	rand	47.50	46.25 ± 1.30	62.36	61.80 ± 0.44	63.02	62.48 ± 0.39	36.63	36.42 ± 0.71
		1600	44.38	45.23 ± 0.97	60.42	60.41 ± 0.59	61.98	61.36 ± 0.40	35.46	35.41 ± 0.50
		160	16.87	16.89 ± 0.73	28.30	22.07 ± 4.03	47.22	47.24 ± 0.44	17.97	17.81 ± 0.55
		16	1.25	1.28 ± 0.36	0.00	3.22 ± 1.51	32.99	32.57 ± 0.83	5.99	6.78 ± 0.48
	1	rand	53.75	53.20 ± 0.54	64.93	64.76 ± 0.23	64.90	64.79 ± 0.17	41.12	41.08 ± 0.43
		1600	52.50	53.01 ± 0.59	64.93	64.51 ± 0.29	64.93	64.86 ± 0.16	40.79	40.84 ± 0.40
		160	30.63	29.04 ± 0.92	50.35	47.92 ± 1.77	54.51	54.41 ± 0.50	26.80	26.95 ± 0.46
		16	3.75	3.11 ± 0.53	0.00	3.86 ± 1.84	34.72	34.62 ± 0.71	7.49	7.96 ± 0.53

Tabla 4.8: Resultados del test de Holm ($\alpha = 0{,}05$) usando $\lambda = 16$ como parámetro de control.

i	estrategia	z-statistic	p-valor	α/i
1	$\lambda = 160$	2.598e+00	4.687e–03	5.000e–02
2	$\lambda = 1600$	7.015e+00	1.151e–12	2.500e–02
3	rand	8.747e+00	1.097e–18	1.667e–02

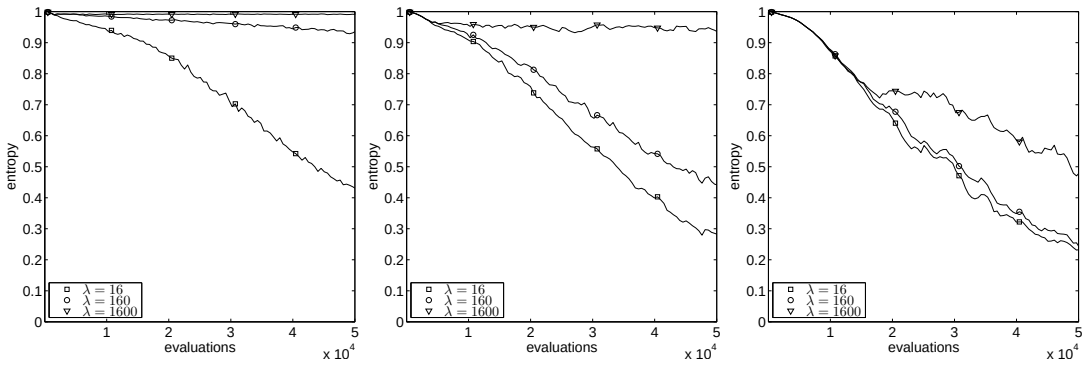
4.3.3. Discusión

Cualquier algoritmo desplegado sobre un entorno computacional inestable debe ser capaz de funcionar adecuadamente en un sustrato tan volátil. Las me-



(a) (b) (c)

Figura 4.15: Evolución del mejor fitness para TRAP con topología SF. De izquierda a derecha: $k = 1$, $k = 5$ y $k = 20$.



(a) (b) (c)

Figura 4.16: Evolución de la diversidad genética para TRAP con topología SF. De izquierda a derecha: $k = 1$, $k = 5$ y $k = 20$.

taheurísticas no son una excepción y aunque tradicionalmente han tenido una adaptabilidad inherente hacia algunas extensiones [212], su diseño debe incorporar políticas adecuadas para enfrentarse con la pérdida de información asociada con los nodos de cómputo que pasan a estar inactivos. Una técnica clásica de tolerancia a fallos diseñada para este propósito es la creación de puntos de restauración periódicos con el estado completo de todos los nodos con el objetivo de poder recuperar el sistema (parcial o completamente) a un estado anterior cuando se produzca algún fallo. En este sentido y en el contexto de los MMAs basados en islas se ha podido comprobar que la estrategia denominada checkpoint puede ser oportuna en escenarios con tasas de *churn* bajas, donde ni siquiera puede ser necesario realizar frecuentes grabaciones del estado de los nodos para obtener un rendimiento apropiado. Sin embargo, en escenarios caracterizados por tasas de *churn* mayores se requieren backups mucho más frecuentes para poder hacer frente a la volatilidad de los nodos. Este sobrecoste adicional de tales backups junto

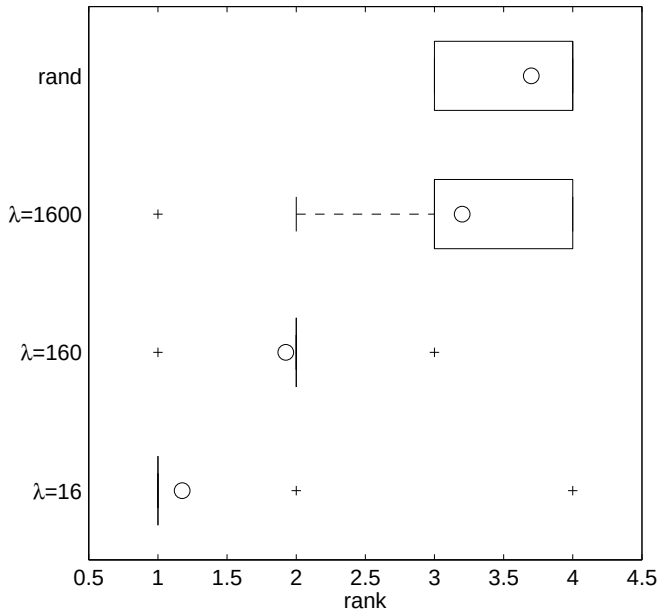


Figura 4.17: Distribución de rankings de las estrategias de reactivación. Cada caja se compone del primer al tercer cuartil de la distribución, de la mediana (cuartil 2) que se identifica mediante una línea vertical, de la media representada con un círculo y de los datos con valores extremos que se indican con un signo más. Las líneas horizontales tienen una amplitud de 1.5 veces el rango entre cuartiles.

con la necesidad de tener acceso a dispositivos de almacenamiento externos persistentes hace que esta aproximación sea menos atractiva en tales situaciones, lo que sugiere que otras aproximaciones –autónomas, auto-adaptativas o puramente locales– pueden ser más apropiadas.

Capítulo 5

Auto-Escalado y Auto-Reparación en el Modelo de Islas

En el capítulo anterior se expusieron algunos mecanismos para desarrollar MMAs sobre entornos distribuidos inestables con diferentes tasas de volatilidad y con topologías de interconexión complejas utilizando diversos medios para mitigar el efecto que puedan tener dichos entornos en el rendimiento de los algoritmos. Esto efecto viene bien representado a través del fenómeno del *churn*, el cual puede tener diferentes consecuencias sobre una metaheurística distribuida basada en población. El caso de los modelos basados en islas, esto es, la distribución de poblaciones separadas sobre diferentes nodos de cómputo sujetas a un intercambio esporádico de soluciones entre ellas, es un claro ejemplo de que el *churn* puede influir en el comportamiento del algoritmo [172]. En general, esto afecta al progreso del proceso de búsqueda debido a que implica la pérdida de información cuando un nodo desaparece del sistema y la introducción de ruido cuando un nuevo nodo vuelve a entrar en el mismo si se reinicializa desde cero. Para tratar con esto hay dos posibilidades: (i) utilizar alguna política de gestión de fallos (e.g., usar computación redundante o puntos de restauración) con propósitos correctivos para los nodos que fallan o se reactivan [212, 227], como se vio en la Sección 4.2 y (ii) dotar al algoritmo de mecanismos de detección de fallos para reaccionar ante el fenómeno del *churn*. Cada aproximación tiene sus propias ventajas e inconvenientes. Por ejemplo, las estrategias de gestión de fallos tales como checkpoint son bastante simples y bien conocidas en el contexto de la computación distribuida. Estas requieren algún tipo de monitorización global y/o la disponibilidad de algún medio de almacenamiento externo de los estados computacionales. En cambio, si el algoritmo puede auto-adaptarse al *churn* (e.g., variando dinámicamente el tamaño de las poblaciones) no se requiere ningún control central ni almacenamiento externo. Esta auto-adaptación permite al algoritmo modificar su comportamiento en función de los recursos computacionales disponibles, lo que hace que el algoritmo cumpla la propiedad self-⋆ denominada auto-escalado (ver

Sección 2.5.2.8).

En este capítulo se analiza cómo las estrategias de equilibrado de carga entre los nodos pueden compensar la pérdida de las islas existentes o la reactivación de nuevas islas (Sección 5.1). A continuación se profundiza en ello a través del estudio de dos topologías de red más complejas, como son las redes de escala libre (SF) y de mundo pequeño (SW), para verificar el impacto que tienen en el rendimiento del auto-equilibrado (Sección 5.2). Finalmente se incorporan dos mecanismos de auto-reparación para compensar este deterioro. En primer lugar el auto-muestreo a través de un modelo probabilístico dinámico sobre las poblaciones de los nodos (Sección 5.3) y en último lugar el auto-cableado representado mediante un procedimiento de reconexionado de los nodos en función de las desconexiones que se van produciendo en la red cuando un nodo abandona el sistema (Sección 5.4). El capítulo finaliza uniendo todas estas propiedades self-$\star$ sobre un mismo MMA dando lugar a un modelo con auto-equilibrado, auto-muestreo y auto-cableado (Sección 5.5).

5.1. Algoritmos Multimeméticos con Equilibrado de Carga

En un EA basado en islas ejecutado sobre una plataforma computacional inestable la volatilidad de los recursos disponibles puede hacer que se pierda la mejor solución (cuando la isla que lo contiene abandona el sistema antes de que el mejor individuo tenga la oportunidad de migrar a otra isla [172]) y ocasionar efectos negativos sobre la diversidad de la población. Se pueden utilizar medidas preventivas para minimizar su impacto y una de ellas es dotar al algoritmo de capacidades de auto-adaptación en función de los fallos que van sucediendo, intentando mantener la diversidad genética y memética durante todo el tiempo. Este tipo de medidas preventivas pueden tener la ventaja de ser inherentemente autónomas y descentralizadas, no siendo necesario conocer el estado global del sistema ni realizar puntos de restauración periódicos.

Esta sección está dedicada a esta última aproximación: se parte del uso de las estrategias de gestión de fallos/recuperaciones consideradas en la Sección 4.2 y se investiga el efecto que tiene la introducción de estrategias descentralizadas de equilibrado de carga sobre el funcionamiento del algoritmo. Con este objetivo, en primer lugar se utiliza un modelo selecto-lamarckiano idealizado –como el visto en la Sección 3.2.1–, el cual permite estudiar algunas cuestiones relacionadas con la diversidad memética y la convergencia. Este modelo se extiende aquí en el contexto de los MMAs basados en islas, como se describió en la Sección 4.1. A continuación se describe un modelo de computación (análogo al utilizado en la Sección 4.2) y se presenta un algoritmo de auto-equilibrado.

Se han utilizado dos estrategias diferentes, una basada en un equilibrado cuantitativo (en la cual las poblaciones ajustan su tamaño dinámicamente para hacer frente a los fallos/recuperaciones de los nodos) y otra basada en un equilibrado

Algoritmo 7: Modelo Selecto-Lamarckiano Basado en Islas

```
for i ∈ [1 ··· n_ι] do in parallel
    Inicializar(pop_i) ;                    // Inicializar población i-ésima
    buffer_i ← ∅ ;                  // inicializar buffer de migración i-ésimo
end
while ¬ CuotaConsumida() do
    for i ∈ [1 ··· n_ι] do in parallel
        ChequearMigrantes (pop_i, buffer_i) ;          // aceptar migrantes
        // ----------Comienzo fase selecto-lamarckiana ---------
        k ← rand(1, μ_i) ;                  // seleccionar posición aleatoria
        ⟨g, m⟩ ←Selección(pop_i, S_i, k) ;          // selección por torneo
        g' ← f(g, m) ;                                    // mejora local
        pop_i ← Reemplazo(pop_i, S_i, k, ⟨g', m⟩);          // del peor padre
        // -----------Fin fase selecto-Lamarckiana -------------
        if rand() < p_mig then
            for j ∈ N_i do
                EnviarMigrantes(pop_i, buffer_j) ;        // envío migrantes
            end
        end
    end
end
```

cualitativo (en la que la información genética y memética se transmite entre los nodos durante el proceso mismo de equilibrado de carga). Estas estrategias se evalúan sobre topologías de red de escala libre y se comparan con una estrategia sin equilibrado de carga que mantiene el tamaño de las islas constante. Inicialmente el interés se va a centrar en el proceso de absorción memético (utilizando para ello el modelo selecto-lamarckiano) y posteriormente se abordará su estudio sobre MMAs completos ejecutados para un conjunto de problemas.

5.1.1. El Modelo Selecto-Lamarckiano Basado en Islas

Como se estableció previamente, la primera parte de la experimentación se ha realizado utilizando un modelo selecto-lamarckiano para obtener una idea preliminar sobre el comportamiento de los MMAs en términos de convergencia y diversidad cuando estos se despliegan sobre un escenario computacional dinámico e inestable. Este modelo es una caracterización abstracta de los MMAs como se definió en la Sección 3.2.1.

Este paradigma básico se extiende ahora con una configuración multipoblación [52] como se muestra en el Algoritmo 7: se consideran n_ι islas que trabajan en paralelo (interconectándolas en función de una cierta topología $\mathcal{N}$), realizándose la fase de migración antes/después de la fase selecto-lamarckiana. La migración

se ejecuta de forma asíncrona: las islas chequean al comienzo de cada ciclo si han recibido o no migrantes. En caso afirmativo, los aceptan en la población siguiendo una política de reemplazo de los migrantes determinada. Posteriormente, al final de cada ciclo, la migración se ejecuta estocásticamente al igual que cualquier otro operador evolutivo (con una cierta probabilidad p_{mig} en este caso). Una vez hecho, algunos migrantes se seleccionan usando una política de selección de migrantes determinada y se envían a las islas vecinas. Siguiendo el análisis de resultados de la Sección 4.1, se utiliza una política de selección de migrantes aleatoria y de reemplazo determinista sustituyendo a los peores individuos de la isla receptora. Este modelo selecto-lamarckiano se puede extender inmediatamente a un MMA completo como el definido en la Sección 3.1, donde los memes se incorporan a los individuos como reglas cuya aplicación altera el genotipo de los mismos.

5.1.2. Estrategias de Auto-equilibrado

La inestabilidad del sistema ocasiona que algunas islas desaparezcan cuando un nodo se cae y por otro lado, se puedan (re-)crear nuevas islas cuando un nodo vuelve a estar disponible. Esto significa que en ausencia de una estrategia para tratar este fenómeno, el tamaño de la población global fluctuará (posiblemente de forma amplia, dependiendo de la tasa de *churn*) afectando a la diversidad genética y memética. Para poder gestionar adecuadamente este fenómeno se introduce una estrategia de equilibrado de carga. Dado el interés inherente a este trabajo por utilizar políticas descentralizadas se considera una estrategia local, tanto en el sentido de toma de decisiones como en cuanto a las políticas de migración, es decir, tanto la elección de las decisiones en cada momento como el intercambio de información se realizan localmente entre las islas vecinas, sin necesidad de disponer de información global del sistema o de un control central [230]. Más concretamente, se usa una variación de la política de vecinos directos[1] [370] como se ilustra en los Algoritmos 8–10.

El procedimiento de equilibrado se activa al comienzo de cada ciclo evolutivo en la isla correspondiente. El mecanismo de adaptación del tamaño de las islas funciona de la siguiente manera:

1. Se incrementa el tamaño de las islas para compensar la pérdida de otras islas.

2. Se equilibran los tamaños de las poblaciones de las islas vecinas.

El núcleo de esta política se captura en el Algoritmo 9: en él dos nodos se comunican e intentan llegar a un estado de equilibrio local entre ellos; si la diferencia entre sus tamaños de población es inferior a un umbral determinado δ, entonces modifican el tamaño de sus poblaciones hasta encontrar un punto de equilibrio (eventualmente alcanzando un equilibrio global también después de

[1] *direct-neighbor policy*

Algoritmo 8: Procedimiento de Equilibrado Estándar

```
procedure Estándar-LB (↓𝒩, ↕A[], n[], W[])
// 𝒩: Lista de referencias a islas vecinas
// A: Vector lógico que mantiene una pista de los vecinos
//    activos.
// n, W: Vectores de enteros con el número de vecinos activos
//       y los tamaños de la población de cada vecino en 𝒩.

for v ∈ 𝒩 do
    if v.ping() then
        // El vecino está activo. Se intenta el equilibrado.
        (n_v, W_v, b) ← Hacer-LB (v);
        A_v ← true;
    else
        if A_v then
            // El vecino estuvo activo la última vez.
            // Se incrementa la población propia.
            w ←ObtenerTamañoPoblación();
            AsignarTamañoPoblación(w + W_v/n_v);
            A_v ← false;
        end
    end
end
```

Algoritmo 9: Rutina Básica de Equilibrado

```
function Hacer-LB (↕v) returns (ℕ, ℕ, 𝔹)
// v: vecino con el que se realiza el equilibrado.
w ←ObtenerTamañoPoblación();
w' ← v.ObtenerTamañoPoblación();
Δ ← w − w';
if |Δ| > δ then
    AsignarTamañoPoblación(w − Δ/2);
    v.AsignarTamañoPoblación(w' + Δ/2);
    b ← true;
else
    b ← false;
end
return (v.ObtenerVecinosActivos(), v.ObtenerTamanoPoblacion(), b)
```

un número determinado de iteraciones [47]). Esta rutina básica se utiliza dentro del procedimiento estándar de equilibrado que se ejecuta en cada nodo (ver Algoritmo 8), por medio del cual se comprueba la disponibilidad de los vecinos

Algoritmo 10: Procedimiento de Equilibrado para Reactivación

```
procedure Reactivación-LB (↓𝒩, ↑A[], n[], W[])
// Los parámetros tienen el mismo significado que en el
// Algoritmo 8.
equilibrado ← ObtenerTamañoPoblación() > 0;
for v ∈ 𝒩 do
    if v.ping() then
        // El vecino está activo. Se intenta el equilibrado.
        (n_v, W_v, b) ← Do-LB (v);
        A_v ← true;
        equilibrado ← equilibrado ∧ b;
    else
        A_v ← false;
    end
end
if ¬equilibrado then
    // No se ha realizado el equilibrado.
    // Reinicialización desde cero.
    AsignarTamañoPoblación(C_1);
end
```

para determinar si están activos o no. Si lo están, se intenta un equilibrado de la carga con ellos. En el caso en que un vecino acabe de abandonar el sistema (i.e., ese nodo estaba activo en el intento previo de equilibrado, pero ha dejado de estarlo), la isla incrementa el tamaño de su propia población en una fracción del tamaño de la población del nodo vecino (proporcional al número de vecinos activos del nodo que ha caído). La situación es ligeramente diferente cuando un nodo se reactiva: intenta el equilibrado con las islas vecinas y si no puede hacer esto (porque no tiene vecinos activos o porque las diferencias entre su población y la del vecino están por debajo del umbral de equilibrado), el nodo recurre a una auto-reinicialización usando un tamaño de población fijo C_1 –ver Algoritmo 10. Considérese que un nodo reactivado podría haber sido el extremo pasivo de un intento de equilibrado de otro nodo antes de entrar en su propio procedimiento de equilibrado, con lo cual podría tener un tamaño de población diferente de cero en el inicio de este procedimiento. En este caso no se requiere una reinicialización del nodo. Finalmente, hay que tener en cuenta que es posible que la red se desconecte en algún momento y por lo tanto, un nodo podría caerse sin tener vecinos activos para absorber una parte de su tamaño de población. En esta situación, el tamaño total de la población podría decrecer eventualmente. Esta es una cuestión que se ha decidido no compensar por las siguientes dos razones: por un lado, es una situación poco probable en escenarios con tasas de *churn* bajas; y por otro lado, su ocurrencia en escenarios con tasas de *churn* elevadas puede

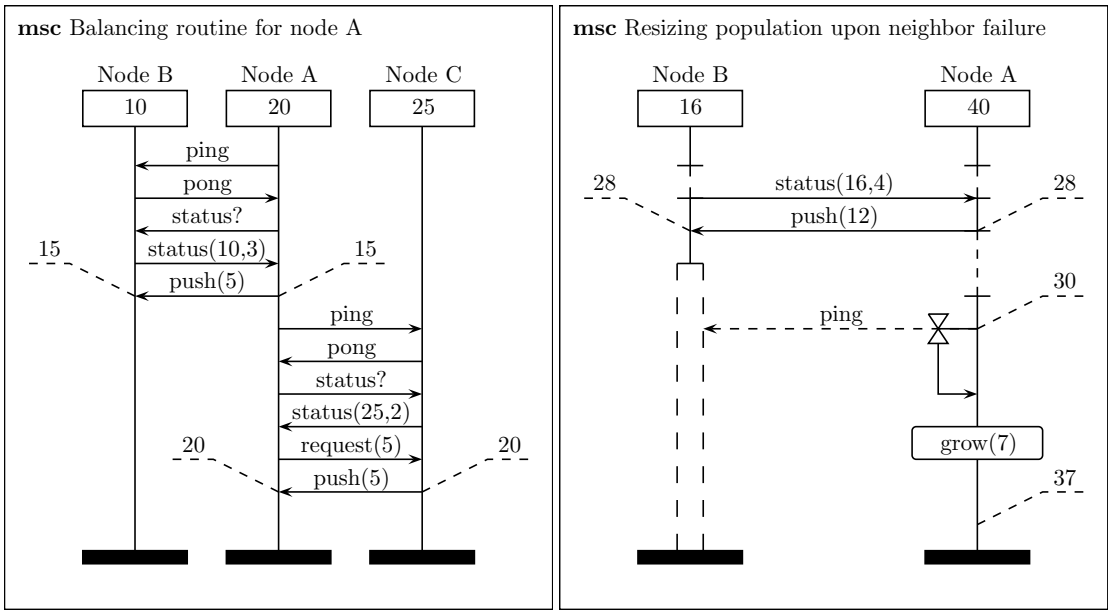


Figura 5.1: Gráfico de secuencia de mensajes del protocolo de equilibrado. (a) Protocolo de equilibrado estándar. Un nodo A se comunica con sus dos vecinos e intenta equilibrar su población con ellos. (b) Ajuste de la población cuando un nodo vecino ha fallado. Un nodo A intenta equilibrarse con un nodo B y se da cuenta de que este ha caído basándose en que ha expirado el tiempo del mensaje *ping*. Entonces incrementa su propia población usando la información recopilada de B durante su último intercambio.

proporcionar información realmente interesante sobre la resistencia inherente de estas técnicas/estrategias.

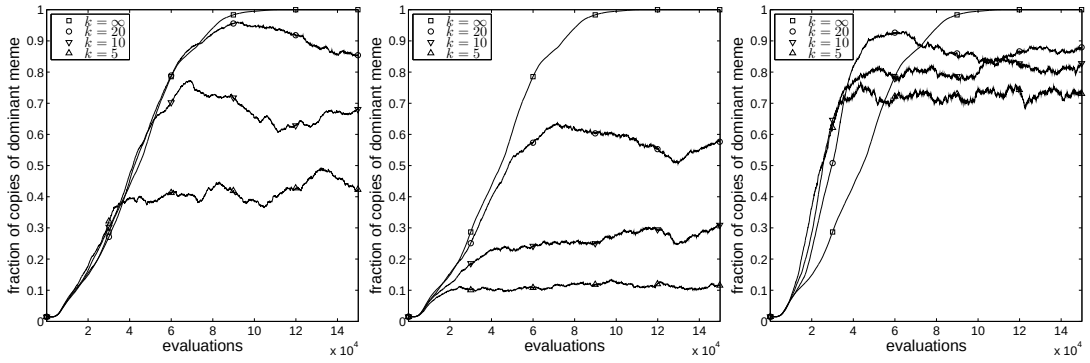
Un ejemplo concreto se muestra en la Figura 5.1. El protocolo básico de equilibrado se ilustra en el gráfico de secuencia de mensajes de la Figura 5.1(a). Un nodo A determinado (cuyo tamaño de población inicial es 20) se comunica con sus vecinos (B y C), haciéndoles *ping* para chequear si están activos y solicitándoles información sobre sus tamaños de población y el número de vecinos activos (el nodo B responde que su tamaño es 10 y tiene 3 vecinos y el nodo C devuelve un tamaño de 25 individuos con 2 vecinos). Con esta información (la cual se almacena en una memoria local para uso posterior) la población se agranda o se contrae para conseguir el equilibrado local (esto es, se calcula el punto medio entre el tamaño de los nodos involucrados y se asigna este tamaño a las poblaciones). En este caso el punto medio entre el nodo A y el nodo B es 15 individuos, por lo que se transfieren 5 individuos (seleccionados de forma aleatoria de la población correspondiente) del nodo con más población (el nodo A) al de menos para alcanzar el equilibrio. Al finalizar este proceso ambos nodos tienen 15 individuos cada uno. A continuación el nodo A repite este mismo proceso con el nodo C, finalizando

ambos con 20 individuos cada uno (obsérvese que al iniciar este último proceso la poblacion del nodo A ya no era 20, sino 15 individuos). En caso de detectar que algún vecino acaba de pasar a un estado inactivo (i.e., estaba activo en el intento previo de equilibrado pero no en el actual), la isla incrementa su propia población para compensar la pérdida de la isla vecina, como se ilustra en la Figura 5.1(b): el nodo supone que la población perdida se distribuye cuantitativamente entre sus vecinos, usando para ello la información recopilada durante la última comunicación con éxito con la isla que está ahora inactiva sobre su número de vecinos activos y el último tamaño de población observado. En este caso el nodo B había informado al nodo A que tenía 28 individuos (los 16 originales más los 12 que recibió previamente) y 4 vecinos. Por ello, A incrementa su población (usando inmigrantes aleatorios, es decir, generando nuevas soluciones aleatorias e insertándolas en la población) en la cantidad correspondiente, que en este caso es $28/4 = 7$, por lo que el nodo A finaliza el equilibrado con 37 individuos en su población.

Este procedimiento de equilibrado descrito en los párrafos anteriores se ha abordado de dos formas diferentes: cuantitativamente y cualitativamente. En la versión cuantitativa el cambio en el tamaño de la población se realiza truncando (eliminando los peores individuos de la población, tantos como se precise) para reducir la población y añadiendo inmigrantes aleatorios [162] (tantos como sea necesario) para agrandarla. Por consiguiente, el equilibrado se hace solo en términos numéricos. En cambio, en la versión cualitativa, el equilibrado conlleva el intercambio de información genética y memética: un paquete de individuos del tamaño deseado se selecciona aleatoriamente en la isla donante y se transfiere a la isla receptora, eliminándose de la primera. Considérese que tanto la reinicialización de las islas (que se realiza desde cero) como la extensión de la población cuando un vecino se cae del sistema son siempre procedimientos cuantitativos en cualquiera de los casos.

5.1.3. Análisis Experimental

Los experimentos con el modelo selecto-lamarckiano se han realizado usando el modelo descrito en la Sección 5.1.1 con $n_\iota = 64$ islas de $\mu = 50$ individuos inicialmente. Cada uno de estos individuos $\langle g_i, m_i \rangle$ se inicializa seleccionando $g_i \sim U(0, 1/2)$ y $m_i \sim U(0, 1)$, así los genotipos tienen la posibilidad de mejorar con memes de alto potencial y se minimizan las posibilidades de que memes de baja calidad prosperen asociándose a genotipos de alta calidad (ver Sección 3.2). El meme se aplica utilizando una combinación lineal $f(g, m) = \gamma g + (1 - \gamma)m$ (para $m > g$). Se ha utilizado $\gamma = 0{,}9$ (i.e., la distancia entre el gen y el meme se decrementa un 10 % en cada aplicación del meme) para tener una curva de mejora moderada. Las variantes algorítmicas con equilibrado cuantitativo y cualitativo se denominan como LB y LBQ respectivamente. Con el fin de establecer comparaciones también se ha utilizado una variante sin equilibrado denominada noB, en la cual los nodos reactivados se reinicializan desde cero. El parámetro m



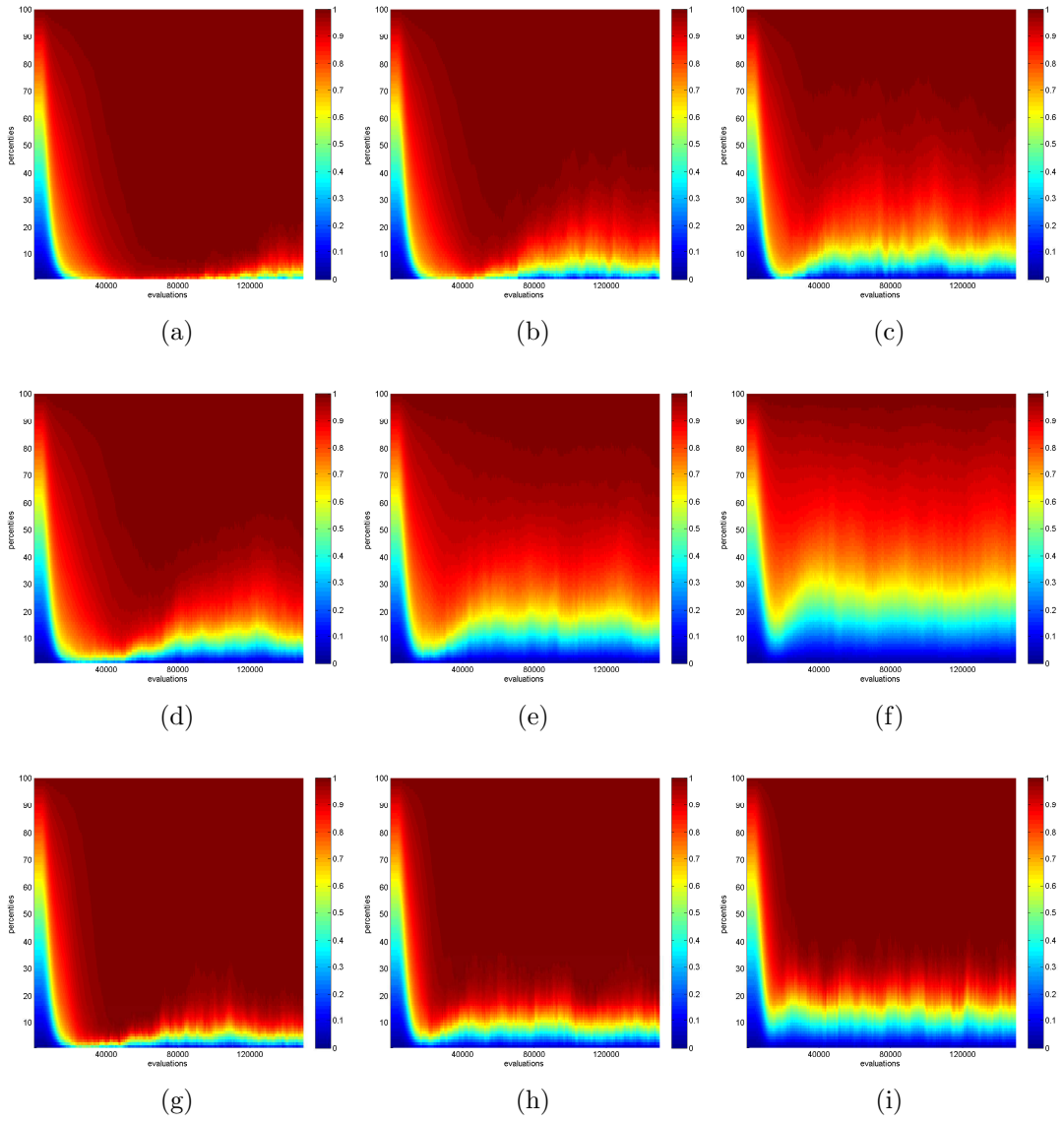
(a) (b) (c)

Figura 5.2: Fracción de copias del meme dominante. De izquierda a derecha: sin equilibrado, equilibrado cuantitativo y equilibrado cualitativo. La curva para $k = \infty$ es la misma para todos los algoritmos.

en el modelo de Barabási-Albert se ha fijado a $m = 2$, con $p_{mig} = 1/250$. Con respecto a la reactivación/desactivación de los nodos se utiliza el parámetro de forma $\eta = 1{,}5$, para tener una tasa de riesgo creciente, y el parámetro de escala $\beta = -1/\log(p)$ con $p = 1 - (kn_\iota)^{-1}$, para $k \in \{5, 10, 20, \infty\}$. El umbral de equilibrado se ha establecido como $\delta = 1$ y se ha utilizado un valor para el parámetro C_1 igual a $2\mu = 100$ individuos, durante una eventual reinicialización de las islas desde cero, dado que el número medio de islas activas con los parámetros utilizados se aproxima asintóticamente a $n_\iota/2$. Se han realizado 25 simulaciones para cada algoritmo y cada escenario de *churn*.

En primer lugar, fijémonos en el meme dominante. La Figura 5.2 muestra la fracción de copias de la población completa correspondiente al meme más difundido. Como era de esperar, mientras la población completa converge eventualmente a un estado homogéneo para $k = \infty$, para valores $k < \infty$ se dirige hacia estados semi-estables en los que solo una fracción de la población es absorbida por el meme dominante. Se puede comprobar en la Figura 5.2 (a) que esta fracción semi-estable es cada vez más baja conforme se incrementa la tasa de *churn*, lo que puede explicarse a través de la pérdida continua de islas en estados avanzados de convergencia y por la reintroducción de nuevas islas. Una visión más cualitativa de la situación se proporciona en la Figura 5.3 (a-c). En ella, los memes se representan por tonos de colores (cuanto más frío –azul– es el color peor es el valor del meme), y cada porción vertical de la figura representa la distribución de memes en un cierto momento. El área roja que representa memes de alta calidad comienza a crecer y se estabiliza en un nivel determinado bajo la presión de los memes de peor calidad que se van reintroduciendo en la población (mitad inferior de cada gráfica).

La situación es más clara en el caso del equilibrado cuantitativo, debido a la diversidad adicional que proporciona la introducción de migrantes aleatorios durante el agrandamiento de las poblaciones, como se muestra en la Figura 5.3 (d-f).



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

Figura 5.3: Mapas de memes para las diferentes estrategias de equilibrado. (Fila superior) Sin equilibrado. (Fila intermedia) Equilibrado cuantitativo. (Fila inferior) Equilibrado cualitativo. En cada fila, de izquierda a derecha: $k = 20$, $k = 10$ y $k = 5$.

Esto provoca que el meme dominante únicamente absorba una pequeña fracción de la población completa –ver Figura 5.2 (b). Esto es completamente diferente al comportamiento del equilibrado cualitativo que parece ser una estrategia más robusta, proporcionando un nivel similar de convergencia para cualquier valor de la tasa de *churn* –ver Figura 5.2 (c). De hecho, usar el equilibrado local para la reconstrucción de las islas cuando se reactivan permite mantener el impulso de la búsqueda, redistribuyendo la población existente entre los nuevos nodos sin tener

que recurrir a la reinicialización aleatoria tan frecuentemente como en los casos noB y LB, consiguiendo de esta manera manejar mejor la agitación del sistema.

Con el objetivo de confirmar los patrones de comportamiento observados, ahora hay que centrarse en el MMA completo, como el descrito en la Sección 4.1.1. Considérese que hay $n_\iota = 32$ islas cuyo tamaño inicial es $\mu = 16$ individuos y se usa $p_{mig} = 1/80$ con un número máximo de 50000 evaluaciones. Cada isla sigue un plan reproductivo de estado estacionario. La evaluación y aplicación de los memes se controla por los parámetros $w = 1$, $p_r = 1/9$, $l_{\text{mín}} = 3$ y $l_{\text{máx}} = 9$ análogamente a lo establecido en la Sección 3.3. Se considera una probabilidad de cruce $p_X = 1{,}0$ y una probabilidad de mutación $p_M = 1/\ell$, donde ℓ es la longitud del genotipo. Se han considerado cuatro funciones de test, la función de Deb (TRAP), las funciones H-IFF y H-XOR y el problema MMDP –ver Apéndice A para una descripción más detallada de estas funciones.

La Tabla 5.1 muestra los resultados completos obtenidos. Como puede comprobarse, hay una degradación del rendimiento considerable tanto en noB como en LB conforme k decrece (es decir, conforme se incrementa la tasa de *churn*) –las diferencias con respecto al escenario sin fallos ($k = \infty$) son estadísticamente significativas ($\alpha = 0{,}05$) en todos los casos excepto para la función H-IFF con $k = 20$. Sin embargo, LBQ es mucho más robusto, con una pérdida de rendimiento notablemente inferior para volatilidades crecientes en concordancia con el comportamiento observado en el modelo selecto-lamarckiano. De hecho, las diferencias entre el algoritmo sin fallos y LBQ no son significativas para $k = 20$ en ninguno de los problemas, y en TRAP o MMDP únicamente son estadísticamente significativas en el escenario más volátil ($k = 5$).

Una perspectiva diferente sobre esto se proporciona en la Figura 5.4 (para la función H-IFF). Obsérvese cómo la convergencia de LBQ se ve afectada por la volatilidad creciente de forma mucho más suave que en el caso de noB y LB (la situación es análoga o incluso más clara en favor de LBQ en el resto de problemas considerados). Además, como se muestra en la Tabla 5.1 la superioridad de LBQ sobre noB y LB para un valor determinado de k es casi siempre estadísticamente significativa. Desde un punto de vista global, si se consideran los resultados de cada estrategia como un par $\langle k, \text{problema}\rangle$, el test de Quade indica que al menos una de las estrategias tiene diferencias significativas (p-valor $\simeq 0$) por lo que se ha realizado un test post-hoc (Holm) usando LBQ como estrategia de control. Como se ilustra en la Tabla 5.2, tanto noB como LB pasan el test lo que confirma que LBQ tiene rendimientos significativamente mejores que las otras dos estrategias.

A continuación se va a proceder a estudiar la estrategia LBQ y el efecto que tiene el parámetro δ, que se utiliza como umbral de equilibrado, sobre su comportamiento. Para este objetivo, se han repetido los experimentos para esta estrategia utilizando los siguientes valores $\delta \in \{10, 20, 40\}$. Como se muestra en la Figura 5.5 para el modelo selecto-lamarckiano, incrementando el umbral δ, la absorción por el meme dominante es más lenta y además este se estabiliza alrededor de una pequeña fracción de la población. Esto se puede explicar a través del patrón de difusión de información en LBQ. Este proceso se activa de forma

Tabla 5.1: Resultados (25 ejecuciones) de los diferentes MMAs sobre los cuatro problemas considerados. Se muestra la mediana ($\tilde{x}$), la media ($\bar{x}$) y el error estándar de la media (σ_x). Los símbolos • y ∘ indican si las diferencias numéricas son significativas o no en función del test ranksum de Wilcoxon ($\alpha = 0{,}05$). El primer símbolo del par corresponde a la comparación con $k = \infty$ y el segundo a la comparación con el mejor algoritmo (indicado con ⋆) para el problema correspondiente y valor de k.

		TRAP			H-IFF		
estrategia	k	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$		$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$	
–	∞	32.0	31.8 ± 0.1		576.0	570.2 ± 5.8	
	20	31.6	31.3 ± 0.2	••	576.0	549.6 ± 11.3	∘∘
sin equilibrado	10	30.0	29.6 ± 0.4	••	576.0	496.7 ± 19.7	•∘
	5	22.4	22.4 ± 0.4	••	308.0	332.1 ± 18.6	••
	20	30.8	30.5 ± 0.3	••	576.0	557.1 ± 8.8	∘⋆
cuantitativo	10	27.4	27.4 ± 0.5	••	450.0	464.2 ± 17.9	••
	5	21.2	20.6 ± 0.4	••	266.0	266.6 ± 7.1	••
	20	32.0	31.9 ± 0.1	∘⋆	576.0	546.9 ± 12.1	∘∘
cualitativo	10	32.0	31.7 ± 0.1	∘⋆	576.0	540.3 ± 12.2	•⋆
	5	30.6	30.5 ± 0.2	•⋆	576.0	521.2 ± 15.7	•⋆
		H-XOR			MMDP		
estrategia	k	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$		$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$	
–	∞	408.0	418.9 ± 8.4		23.6	23.5 ± 0.1	
	20	372.0	383.4 ± 7.1	••	23.3	22.9 ± 0.2	••
sin equilibrado	10	343.0	348.8 ± 6.4	••	20.8	20.7 ± 0.2	••
	5	263.0	267.4 ± 5.3	••	17.5	17.4 ± 0.2	••
	20	376.0	370.6 ± 5.8	••	21.8	21.7 ± 0.2	••
cuantitativo	10	317.0	319.7 ± 6.2	••	19.8	19.9 ± 0.2	••
	5	253.0	254.1 ± 4.3	••	16.5	16.7 ± 0.2	••
	20	400.0	407.3 ± 6.8	∘⋆	23.6	23.5 ± 0.1	∘⋆
cualitativo	10	384.0	389.7 ± 7.1	•⋆	23.3	23.2 ± 0.2	∘⋆
	5	371.0	373.2 ± 4.6	•⋆	22.8	22.0 ± 0.2	•⋆

Tabla 5.2: Resultados del test de Holm ($\alpha = 0{,}05$) usando LBQ como algoritmo de control.

i	estrategia	z-statistic	p-valor	α/i
1	noB	2.041e+00	2.061e–02	5.000e–02
2	LB	4.082e+00	2.000e–05	2.500e–02

repentina, por impulsos que resultan de la desactivación/reactivación de las islas: un nodo que se cae del sistema hace que se agranden sus islas vecinas, provocando

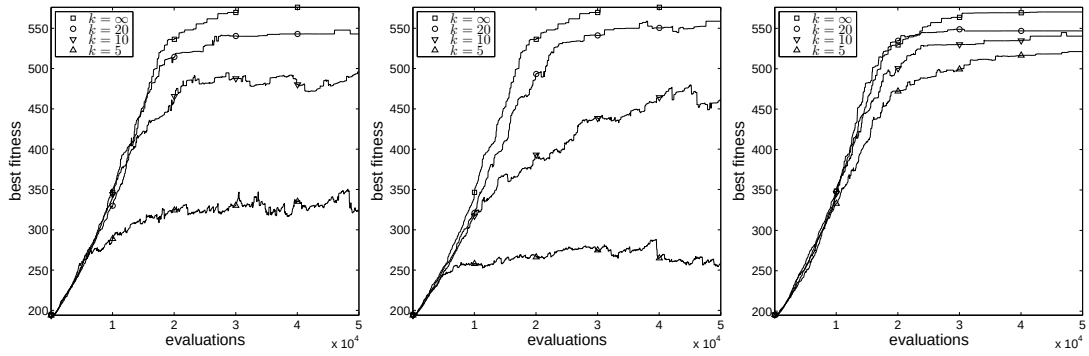


Figura 5.4: Evolución del mejor fitness para la función H-IFF dependiendo del parámetro k. De izquierda a derecha: sin equilibrado, equilibrado cuantitativo, equilibrado cualitativo. La curva para $k = \infty$ es la misma para todos los algoritmos.

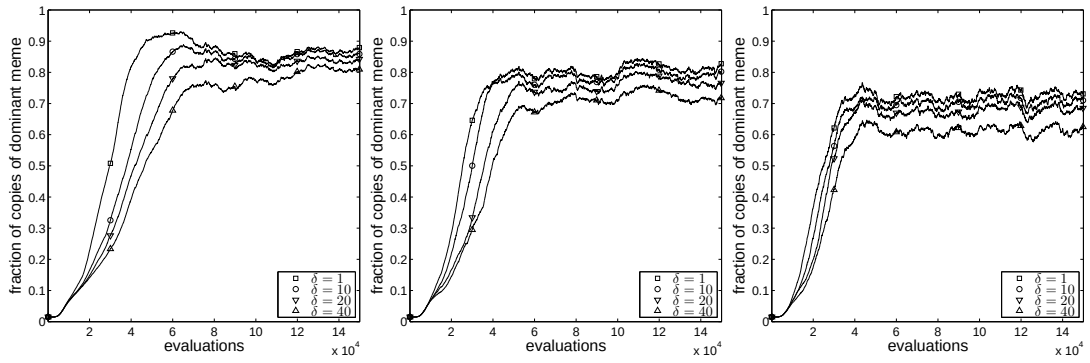


Figura 5.5: Fracción de copias del meme dominante en el modelo selecto-lamarckiano usando la estrategia LBQ con diferentes valores del parámetro del umbral de equilibrado δ (de izquierda a derecha: $k = 20, 10$ y 5).

un flujo de información en el sentido opuesto; por el contrario, cuando un nuevo nodo vuelve a estar disponible provoca que los nodos vecinos le donen parte de sus poblaciones, provocando en este momento un flujo de información hacia él. Cuando el valor del umbral δ es bajo el efecto de estos flujos de información es mayor y al contrario, valores altos de δ introducen un efecto de amortiguación en la propagación de dichos flujos de información.

De la misma forma se ha realizado una experimentación análoga con el MMA completo sobre los mismos problemas anteriores. Los resultados se proporcionan en la Tabla 5.3. Como se puede apreciar, los resultados de LBQ se degradan claramente para los valores más altos del umbral de equilibrado, concretamente para los escenarios más volátiles. Aunque está claro que haciendo que δ crezca se producen menos equilibrados, por lo que asintóticamente (δ muy grande) el rendimiento se degradará al nivel de noB, es también interesante considerar que para un valor moderadamente pequeño de δ, por ejemplo $\delta = 10$, los resultados

Tabla 5.3: Resultados (25 ejecuciones) de la estrategia LBQ para diferentes valores del parámetro del umbral de equilibrado δ sobre los cuatro problemas considerados. Se muestra la mediana ($\tilde{x}$), la media ($\bar{x}$) y el error estándar de la media (σ_x). Los símbolos $\bullet$ y $\circ$ indican si las diferencias numéricas son significativas o no en función de un test ranksum de Wilcoxon ($\alpha = 0{,}05$) con respecto al mejor valor de δ (indicado con $\star$) para el correspondiente problema y k. Los resultados para $\delta = 1$ se toman de la Tabla 5.1 y se incluyen para facilitar la lectura.

		TRAP			H-IFF		
δ	k	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$		$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$	
1	20	32.0	31.9 ± 0.1	$\star$	576.0	546.9 ± 12.1	$\circ$
	10	32.0	31.7 ± 0.1	$\star$	576.0	540.3 ± 12.2	$\circ$
	5	30.6	30.5 ± 0.2	$\star$	576.0	521.2 ± 15.7	$\circ$
10	20	32.0	31.8 ± 0.1	$\circ$	576.0	562.6 ± 7.4	$\star$
	10	31.2	31.2 ± 0.2	$\bullet$	576.0	537.8 ± 13.1	$\circ$
	5	29.8	29.8 ± 0.3	$\circ$	576.0	529.6 ± 15.0	$\star$
20	20	32.0	31.7 ± 0.1	$\circ$	576.0	562.3 ± 8.3	$\circ$
	10	31.2	30.8 ± 0.2	$\bullet$	576.0	535.6 ± 13.9	$\circ$
	5	28.8	28.2 ± 0.4	$\bullet$	576.0	502.5 ± 20.0	$\circ$
40	20	31.6	31.0 ± 0.2	$\bullet$	576.0	543.6 ± 13.7	$\bullet$
	10	29.0	28.8 ± 0.3	$\bullet$	576.0	550.2 ± 12.3	$\star$
	5	23.6	23.6 ± 0.4	$\bullet$	418.0	426.6 ± 23.5	$\bullet$
		H-XOR			MMDP		
δ	k	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$		$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$	
1	20	400.0	407.3 ± 6.8	$\circ$	23.6	23.5 ± 0.1	$\star$
	10	384.0	389.7 ± 7.1	$\star$	23.3	23.2 ± 0.1	$\star$
	5	371.0	373.2 ± 4.6	$\star$	22.8	22.0 ± 0.2	$\star$
10	20	403.0	411.4 ± 7.0	$\star$	23.3	23.0 ± 0.1	$\bullet$
	10	385.0	381.4 ± 4.7	$\circ$	22.2	22.4 ± 0.2	$\bullet$
	5	356.0	358.4 ± 3.8	$\bullet$	21.2	21.4 ± 0.2	$\bullet$
20	20	400.0	406.2 ± 6.3	$\circ$	23.3	22.9 ± 0.1	$\bullet$
	10	370.0	368.8 ± 4.6	$\bullet$	21.8	21.8 ± 0.2	$\bullet$
	5	329.0	329.2 ± 4.8	$\bullet$	20.0	19.9 ± 0.1	$\bullet$
40	20	383.0	384.7 ± 5.4	$\bullet$	22.2	22.3 ± 0.1	$\bullet$
	10	348.0	352.8 ± 5.4	$\bullet$	20.8	20.6 ± 0.2	$\bullet$
	5	286.0	285.0 ± 2.6	$\bullet$	17.8	17.9 ± 0.1	$\bullet$

obtenidos son comparables a los de $\delta = 1$. De hecho, realizando un test estadístico múltiple sobre diferentes valores de δ se observa que no hay diferencias estadísticamente significativas que se puedan establecer entre $\delta = 1$ y $\delta = 10$ –ver Tabla 5.4 (p-valor $\simeq 0$ para el test de Quade). Esto sugiere que la estrategia LBQ es de alguna manera robusta para pequeñas variaciones de este parámetro y que puede haber espacio para un ajuste fino en situaciones específicas.

Tabla 5.4: Resultados del test de Holm ($\alpha = 0{,}05$) usando $\delta = 1$ como algoritmo de control.

i	valor de δ	z-statistic	p-valor	α/i
1	10	7.906e–01	2.146e–01	5.000e–02
2	20	3.004e+00	1.330e–03	2.500e–02
3	40	4.427e+00	<1.000e–05	1.700e–02

5.1.4. Discusión

El desarrollo de estrategias de auto-equilibrado de carga en MMAs sobre entornos inestables tales como las redes P2P o las redes de computación voluntaria requiere que los algoritmos sean capaces de gestionar adecuadamente la volatilidad intrínseca asociada a este tipo de plataformas donde la inestabilidad de los nodos es una realidad. Para ello los algoritmos deben ser robustos y resistentes al fenómeno del *churn*. En este sentido se ha visto que las técnicas de auto-equilibrado son interesantes ya que proporcionan un medio para corregir (o al menos aliviar) estos inconvenientes y además lo hacen de una forma descentralizada, mitigando adicionalmente la perturbación ejercida por este tipo de entornos inestables sobre el proceso de búsqueda del algoritmo.

En primer lugar, los resultados experimentales en el modelo selecto-lamarckiano idealizado de MMAs han permitido corroborar que las dos estrategias de equilibrado de carga consideradas tienen comportamientos complementarios en términos de preservación de la diversidad, aunque los resultados indican que la versión cualitativa es más robusta ante el fenómeno del *churn*. Esto además se confirma a partir de la experimentación sobre MMAs completos donde la versión cualitativa proporciona también el mejor rendimiento en términos globales superando al resto de variantes.

Los resultados que se han obtenido son prometedores en este sentido y sugieren que una estrategia de equilibrado cualitativa puede hacer que los algoritmos sean robustos en este tipo de escenarios. Un estudio más en profundidad sobre el umbral de equilibrado δ permite concluir que el rendimiento de la estrategia de equilibrado cualitativo empeora para valores grandes de δ, pero es robusta para valores más bajos de este parámetro. Si se mira con cierta perspectiva, esto abre las puertas a nuevas vías para el desarrollo de otras estrategias de equilibrado como por ejemplo realizar dinámicamente de forma auto-adaptativa un ajuste más fino del umbral de equilibrado.

5.2. Análisis de Topologías

En esta sección se aborda cómo influye el uso de topologías de red complejas sobre el rendimiento de un MMA basado en islas con auto-equilibrado. El entorno

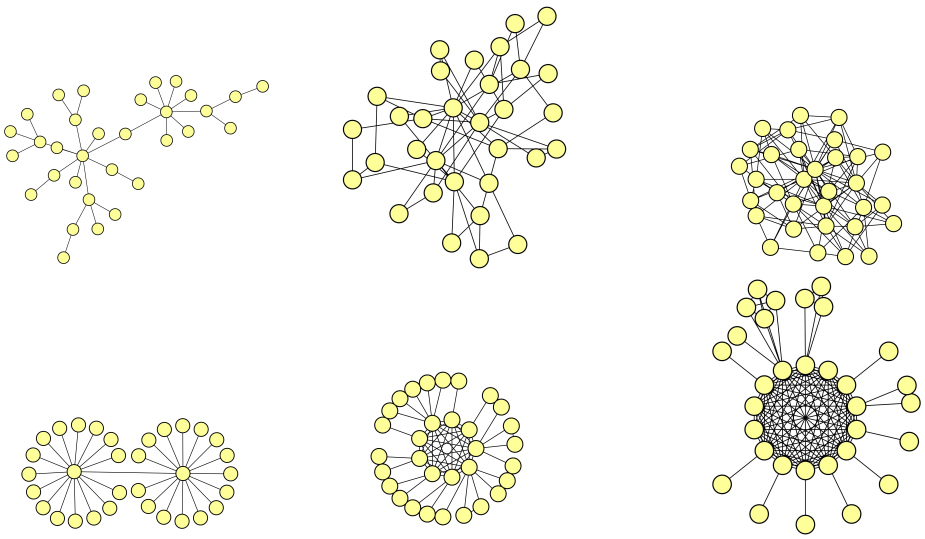


Figura 5.6: Ejemplos de redes. La fila superior corresponde a redes SF y la fila inferior a redes SW. Para cada tipo de red y de izquierda a derecha, los ejemplos corresponden a: $m = 1$, $m = 2$, $m = 4$.

se confecciona como un sistema con islas interconectadas usando una topología compleja no regular, como redes de escala libre y de mundo pequeño.

5.2.1. Modelo Computacional Dinámico

El modelo computacional utilizado es el mismo que el de la sección anterior. Se trata de un sistema distribuido simulado con n_ι nodos y caracterizado por dos elementos fundamentales:

- La topología de interconexión entre los nodos.
- La dinámica de disponibilidades/indisponibilidades de cada nodo.

Con respecto a la topología de interconexión se han considerado dos posibilidades: las redes de escala libre (SF) y las redes de mundo pequeño (SW) –ver Sección 4.3 para más detalles. La Figura 5.6 muestra un ejemplo de ambos tipos de redes. Por otro lado, la dinámica del sistema sigue el modelo presentado en la Sección 4.2 en la que se parte de n_ι nodos inicialmente disponibles y cuya permanencia en el sistema sigue la conocida distribución de Weibull [360]. Al igual que en las secciones precedentes se ha considerado el valor del parámetro de forma de la distribución $\eta > 1$ para que la probabilidad de que un nodo caiga se incremente conforme más tiempo lleve activo en el sistema –ver Sección 4.2.2. Por simplicidad se vuelve a considerar la misma distribución tanto para las reactivaciones de los nodos como para las indisponibilidades.

5.2.2. Análisis Experimental

Se han considerado $n_\iota = 32$ islas cuyo tamaño inicial es $\mu = 16$ individuos y un número total de 50000 evaluaciones. Las longitudes de los memes evolucionan en el rango $l_{\text{mín}} = 3$ y $l_{\text{máx}} = 9$, modificando sus longitudes con una probabilidad $p_r = 1/9$. Se usa una probabilidad de cruce $p_X = 1{,}0$, una probabilidad de mutación $p_M = 1/\ell$, donde ℓ es la longitud del genotipo, la probabilidad de migración es $p_{mig} = 1/80$ y el MMA sigue un plan reproductivo de estado estacionario. El parámetro m en el modelo BA de las redes SF es $m \in \{1, 2, 4\}$ y se utiliza el valor correspondiente de $M = nm - m(m+1)/2$ en el caso del modelo BM de las redes SW. Con respecto a la dinámica de los nodos, se utiliza un parámetro de forma $\eta = 1{,}5$ (por lo que la probabilidad de fallo se incrementa con el tiempo), y el parámetro de escala es $\beta = -1/\log(p(k))$ para $p(k) = 1 - (kn_\iota)^{-1}$, con $k \in \{1, 2, 5, 10, 20\}$. El parámetro C_1 utilizado para una eventual reinicialización de las islas desde cero se considera que es $2\mu = 32$. Se realizan 25 simulaciones para cada algoritmo y escenario de *churn*. Se denominan, al igual que en la Sección 5.1, como noB y LBQ, las variantes algorítmicas sin equilibrado (i.e., no hay un cambio en el tamaño de las poblaciones y la reinicialización es desde cero) y con equilibrado respectivamente, y se utilizan los superíndices ^{SF} y ^{SW} para denotar el uso de topologías SF o SW. Se han considerado cuatro funciones de test: TRAP, H-IFF, H-XOR y MMDP –ver Apéndice A para una descripción de estas funciones.

La Figura 5.7 muestra los resultados obtenidos en términos de la desviación con respecto a la solución óptima (promediados para los cuatro problemas considerados) como función de la tasa de *churn*, por separado para cada densidad de red (los datos numéricos detallados se proporcionan en las Tablas B.9–B.12 en el Apéndice B.2). En primer lugar, es clara la degradación del rendimiento conforme se incrementa la tasa de *churn*. A pesar de esto, se puede observar que las variantes que realizan un equilibrado de carga son notablemente mejores que sus homólogas sin equilibrado. Esto se puede demostrar que es estadísticamente significativo utilizando un test signrank [365] (p-valor = 0,0017 en SW y $< 10^{-14}$ para SF). Esto confirma el trabajo de investigación previo sobre este tipo de estrategias y valida su utilidad sobre diferentes topologías de red.

Una perspectiva más interesante se podría obtener realizando un análisis de la densidad de la red o sobre la clase de red. Empezando con lo último, noB y LBQ ofrecen resultados cualitativamente diferentes. Por un lado, noB^{SW} parece funcionar ligeramente mejor que noB^{SF}, aunque es cierto que la diferencia es bastante pequeña. Por otro lado, LBQ^{SF} proporciona una pequeña mejora con respecto a LBQ^{SW} (estadísticamente significativa para $m = 2, 4$ de acuerdo con signrank, p-valor $< 0{,}001$). Una posible interpretación de esto puede establecerse en términos de la presencia de cliques más grandes en las redes SW. Esto puede facilitar una mejor distribución de las soluciones en escenarios sin equilibrado, mitigando de alguna manera (aunque sea solo minímamente) el efecto del *churn*.

Sin embargo, en presencia de equilibrado estos clusters inducen menos inter-

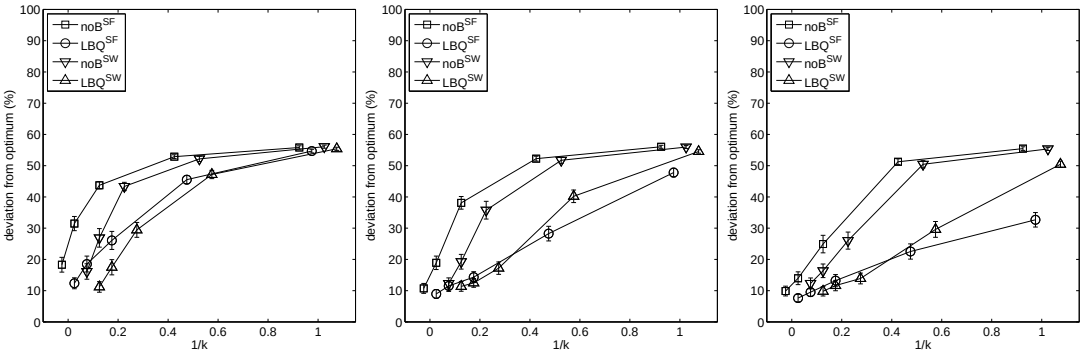


Figura 5.7: Desviación con respecto a la solución óptima representado en función de la tasa de *churn* para $m = 1$ (izquierda), $m = 2$ (central) y $m = 4$ (derecha).

Tabla 5.5: Resultados del test de Holm ($\alpha = 0{,}05$) usando SF_4 como algoritmo de control.

i	estrategia	z-statistic	p-valor	α/i
1	SW_4	2.629e+00	4.280e–03	5.000e–02
2	SF_2	3.406e+00	3.300e–04	2.500e–02
3	SW_2	5.916e+00	$<$ 1.000e–05	1.700e–02
4	SW_1	8.785e+00	$<$ 1.000e–05	1.300e–02
5	SF_1	9.3833+00	$<$ 1.000e–05	1.0003–02

cambios de información que sus equivalentes de las redes SF (una ola de equilibrado creada cuando un nodo se reactiva o se cae puede alcanzar grandes distancias en las redes SF debido al menor grado de los nodos en promedio en este tipo de redes). Esto es consistente con la evolución de los tamaños de las islas mostrados en la Figura 5.8 (primera y segunda columnas). En presencia de altas tasas de *churn* o baja conectividad (i.e., valores bajos de m) la red SF puede llegar a quedarse esporádicamente desconectada. Si no se introducen correcciones adicionales en el procedimiento de equilibrado el resultado puede ser un decremento eventual del tamaño de las poblaciones. Las redes SW son mucho más robustas en este sentido y gestionan adecuadamente el mantenimiento del tamaño de las poblaciones, es decir, consiguen tamaños de población más estables. Esta reducción de los tamaños de población puede verse como un efecto añadido que estimula la convergencia, como se puede ver por ejemplo en la Figura 5.8 (tercera y cuarta columnas) para el caso de la función TRAP. La diferencia entre SW y SF llega a ser más notoria cuando se incrementa la tasa de *churn* y/o decrementa la conectividad, de acuerdo con la hipótesis previa.

Se han analizado también las topologías SF y SW separadamente para cada valor de m. El test de Quade indica que al menos una de las combinaciones $\langle$topología, $m\rangle$ es significativamente diferente (p-valor $\simeq 0$). Se ha realizado también un test post hoc, i.e., el test de Holm usando SF_4 como algoritmo de control. Los resultados de este test se muestran en la Tabla 5.5: el test lo pasan todas las

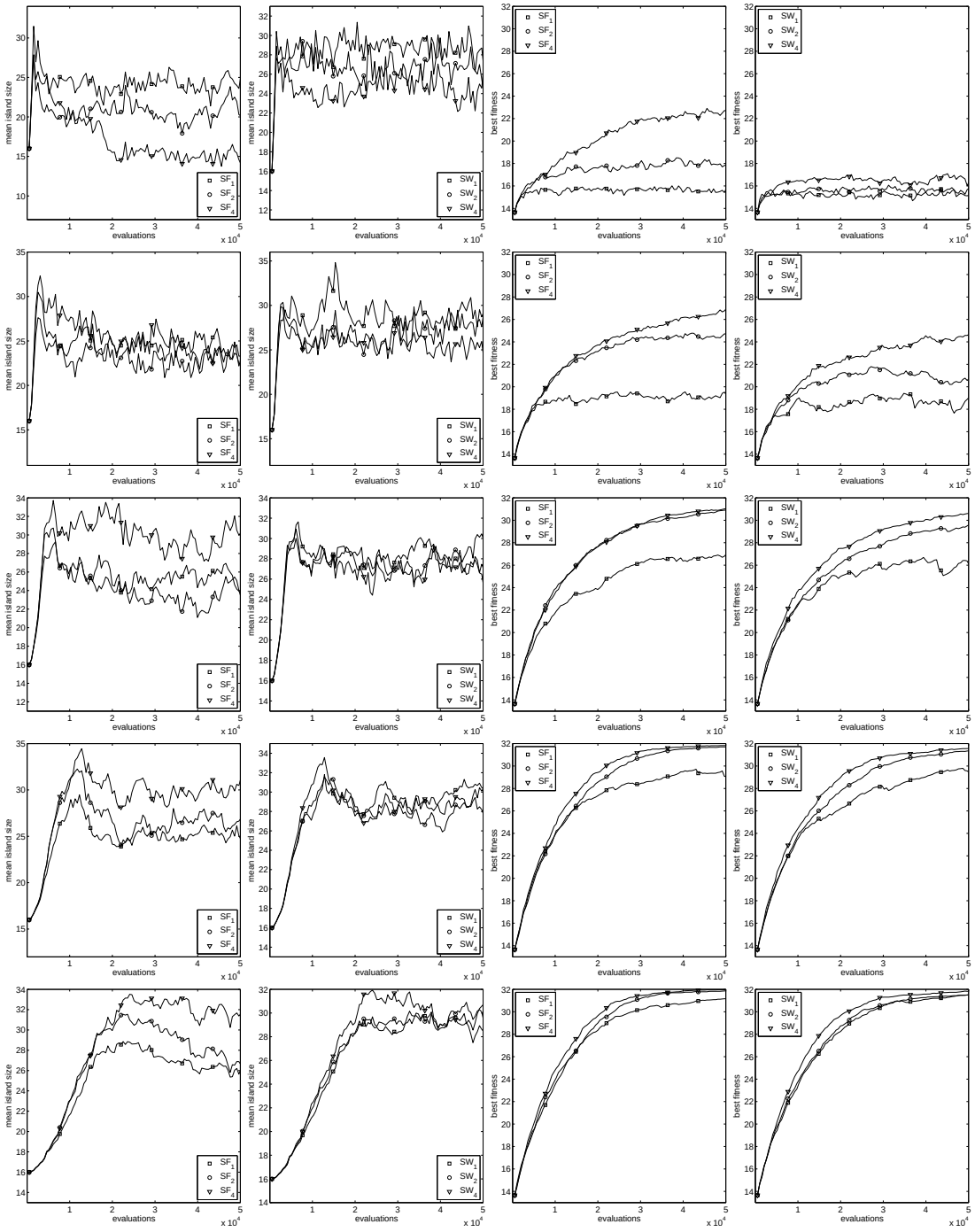


Figura 5.8: Evolución de los tamaños medios de las islas para SF (columna de más a la izquierda) y SW (segunda columna). Evolución del mejor fitness medio para SF (tercera columna) y SW (columna de más a la derecha). De arriba hacia abajo, $k = 1, 2, 5, 10$ y 20.

topologías restantes para todos los valores de m, indicando por lo tanto que esta

topología es la más consistente.

5.2.3. Discusión

La inclusión de topologías de red complejas en MMAs que se ejecutan en entornos inestables puede ser un medio para contrarrestar el efecto negativo que tiene sobre el rendimiento del algoritmo el fenómeno del *churn*. Se ha podido verificar el efecto que tiene el escenario computacional subyacente, caracterizado por una tasa de *churn* determinada, una topología de red concreta y una densidad de conexiones dada. Concretamente, las redes de mundo pequeño con una distancia entre nodos extremadamente reducida y un alto coeficiente de agrupamiento han proporcionado resultados bastante buenos para MMAs tolerantes a fallos con auto-equilibrado, pero son globalmente superadas por las redes de escala libre, particularmente en los escenarios más severos caracterizados por altas tasas de *churn* y/o baja conectividad. También se ha podido verificar que incrementando la densidad de la red el algoritmo es más tolerante a la inestabilidad, sobre todo cuando se combina con el auto-equilibrado, lo que permite deducir que existen sinergias entre estas dos características que dotan al algoritmo de una alta capacidad para sobreponerse a situaciones delicadas donde las tasas de *churn* son elevadas.

5.3. Estrategias de Auto-Muestreo

Esta sección se centra en la influencia que tienen sobre el rendimiento global del algoritmo los mecanismos utilizados para crear nuevos individuos para agrandar las islas activas y compensar la pérdida de otras islas que se ejecutaban en nodos que abandonaron el sistema. Para ello, se considera el uso de modelos probabilísticos de las poblaciones actuales, los cuales a continuación sirven para realizar muestreos con el objetivo de producir soluciones diversas sin distorsionar la convergencia de la población ni el progreso de la búsqueda.

5.3.1. Uso de Modelos Probabilísticos

Las estrategias dinámicas de ajuste del tamaño de la población descritas en la sección 5.1 o bien solo capturan el aspecto cuantitativo del proceso (estrategia LB) o bien utilizan una aproximación elemental para solucionar esta cuestión. La aproximación más sencilla para resolver el aspecto cualitativo de cómo producir los individuos necesarios es recurrir a la búsqueda ciega: las nuevas soluciones –tantas como sea necesario– se crean desde cero, usando básicamente el mismo mecanismo que el utilizado para crear la población inicial. Aunque este puede ser un método apropiado para estimular la diversidad a través de la introducción de individuos nuevos, tiene la desventaja de que no es capaz de mantener el impulso de la búsqueda. De hecho estos nuevos individuos aleatorios pueden ser vistos

como un retroceso de la población en términos de convergencia global, ya que en cierta forma cancelan parte de la exploración realizada durante el proceso hacia otras regiones del espacio de búsqueda. Es obvio que este efecto será más o menos fuerte dependiendo de la frecuencia con la que el algoritmo tiene que activar este mecanismo, lo que estará directamente relacionado con la severidad del *churn*, es decir, con la volatilidad del sistema.

Una alternativa a esta reinicialización aleatoria desde cero es utilizar estrategias más inteligentes basadas en modelos probabilísticos. La idea subyacente sería estimar la distribución de probabilidad que representa la población de la isla que se va a agrandar; a continuación esa distribución se muestrea tantas veces como sea necesario para producir los nuevos individuos. Con esto se persiguen dos objetivos:

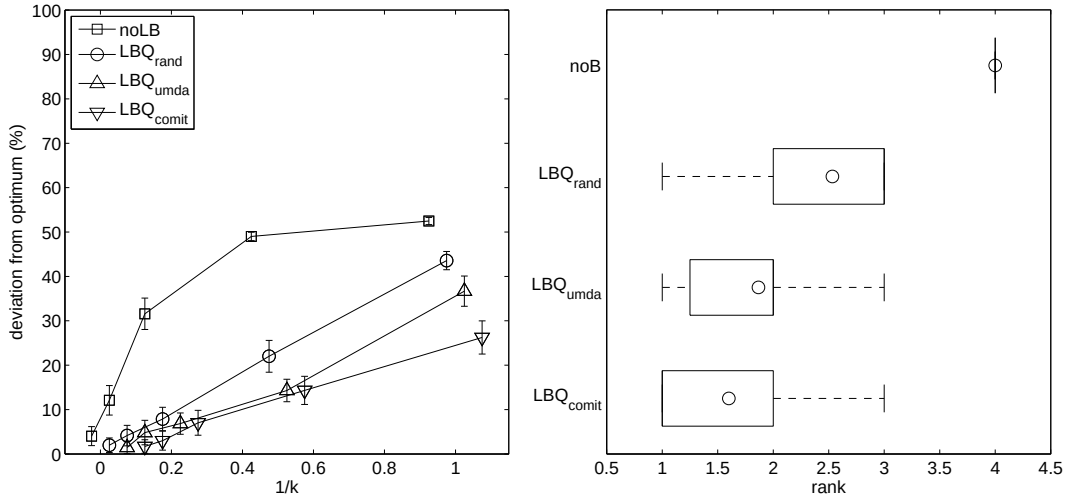
1. El impulso de la búsqueda se mantiene ya que los nuevos individuos son una muestra representativa del estado actual de la población (tal y como se manifiesta a través del modelo probabilístico).

2. La diversidad todavía se mantendría, ya que los nuevos individuos pueden ser diferentes de los existentes en la población.

Para aproximar este modelo probabilístico de la población se han considerado dos alternativas para modelar la población representadas a través de EDAs. Por un lado se utiliza un modelo univariable (UMDA) y por otro uno bivariable (COMIT) tal y como se definieron en la Sección 3.4.

5.3.2. Análisis Experimental

Los experimentos se han realizado usando un MMA con $n_\iota = 32$ islas cuyo tamaño inicial es $\mu = 16$ individuos interconectadas utilizando una topología de escala libre generada con el modelo de Barabási-Albert usando el parámetro $m = 2$. Se considera una probabilidad de cruce $p_X = 1{,}0$, una probabilidad de mutación $p_M = 1/\ell$ –donde ℓ es la longitud del genotipo– y una probabilidad de migración $p_{\text{mig}} = 1/80$. Como en secciones anteriores, los memes son reglas de reescritura siguiendo el modelo presentado en la Sección 3.1.2, cuyas longitudes varían desde $l_{\text{mín}} = 3$ hasta $l_{\text{máx}} = 9$ y se muta su longitud con probabilidad $p_r = 1/l_{\text{máx}}$. Para controlar el *churn*, se considera que la distribución de Weibull que describe la disponibilidad de los nodos se define con el parámetro de forma $\eta = 1{,}5$ (lo que implica tasas de riesgo crecientes ya que este parámetro es mayor que 1) y por seis parámetros de escala $\beta = -1/\log(p)$ para $p = 1 - (kn_\iota)^{-1}$, con $k \in \{1, 2, 5, 10, 20, \infty\}$ que describen escenarios con diferente volatilidad: desde no volatilidad ($k = \infty$) a altas tasas de *churn* ($k = 1$). El parámetro C_1 utilizado durante una eventual reinicialización de las islas desde cero se ha activado con el valor $2\mu = 32$. Se han realizado 25 simulaciones para cada algoritmo y escenario de *churn*, cada una de ellas constando de 50000 evaluaciones. Los diferentes MMAs se denominan LBQ_{rand}, LBQ_{umda} y $\text{LBQ}_{\text{comit}}$ según usen reinicialización aleatoria



(a) (b)

Figura 5.9: (a) Desviación media con respecto a la solución óptima para los tres problemas considerados y para cada algoritmo en función de la tasa de *churn*. (b) Distribución de rankings para cada algoritmo. Como es habitual cada caja se compone del primer al tercer cuartil de la distribución, de la mediana (cuartil 2) que se identifica mediante una línea vertical, de la media representada con un círculo y de los datos con valores extremos que se indican con un signo más. Las líneas horizontales tienen una amplitud de 1.5 veces el rango entre cuartiles.

de los individuos o modelos probabilísticos univariables (UMDA) o bivariables (COMIT) respectivamente. A efectos de poder establecer comparaciones que sirvan como referencia, se ha considerado adicionalmente un MMA sin equilibrado de carga que se ha denominado como noB. Se han considerado tres funciones de test: TRAP (concatenando 32 trampas de cuatro bits cada una), H-IFF (usando 128 bits) y MMDP (usando 24 bloques de seis bits cada uno) –ver el Apéndice A para más detalles.

Una perspectiva global de los resultados se proporciona en la Figura 5.9(a). Esta figura muestra la distancia media al valor óptimo obtenido por cada algoritmo –promediado para los tres problemas– en función de la tasa de *churn* (los resultados numéricos completos se proporcionan en la Tabla 5.6). No es sorprendente que la distancia al óptimo crezca conforme se incrementa la tasa de *churn* en todos los algoritmos. Sin embargo, sí es interesante observar cómo los diferentes algoritmos hacen esto de manera diferente. En el caso de noB hay una degradación del rendimiento muy abrupta conforme k decrece (es decir, conforme se incrementa el *churn*; obsérvese que el eje X de la gráfica es $1/k$, lo que implica que desplazamientos hacia la derecha en el eje X incrementan el valor de $1/k$, pero disminuyen el de k), pero esta degradación es mucho más suave (incluso lineal) para las variantes con equilibrado de carga. Entre estas, las variantes con el modelo probabilístico tienen un rendimiento inequívocamente mejor que

Tabla 5.6: Resultados (25 ejecuciones) en términos de la desviación con respecto a la solución óptima de los diferentes MMAs sobre los tres problemas considerados. Se muestra la mediana ($\tilde{x}$), la media ($\bar{x}$) y el error estándar de la media ($\sigma_{\bar{x}}$). Los tres símbolos próximos a cada entrada indican si las diferencias son estadísticamente significativas (•) o no lo son (○). El primer símbolo corresponde a una comparación entre el algoritmo correspondiente y la versión sin fallos ($k = \infty$); el segundo refleja una comparación con LBQ_{rand} y el tercero es una comparación con el algoritmo que proporciona el mejor resultado para el correspondiente problema y valor de k (identificado mediante ⋆).

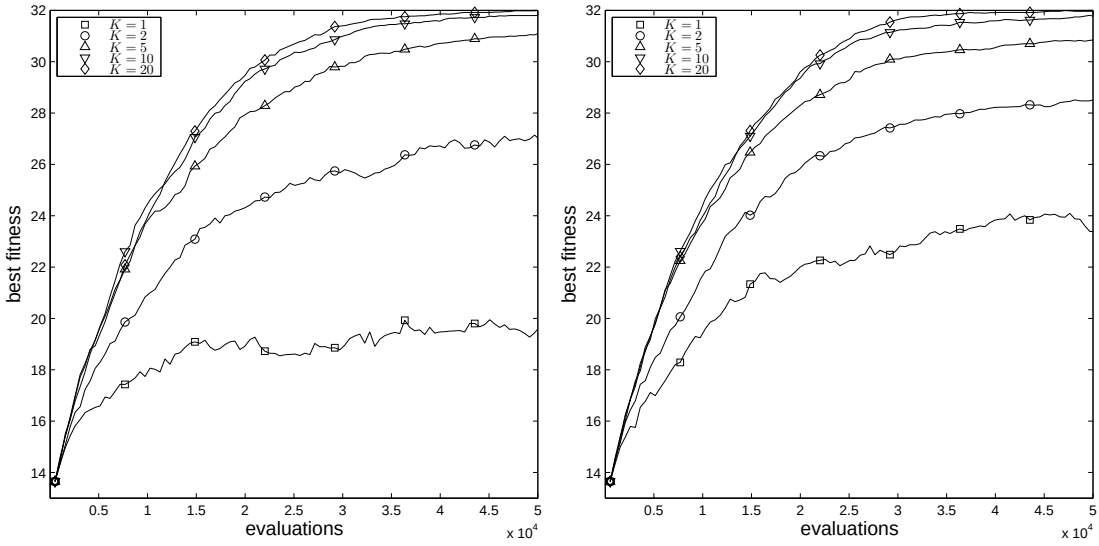
		TRAP			H-IFF			MMDP		
estrategia	K	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$		$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$		$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$	
–	∞	0.00	0.55 ± 0.18		0.00	1.00 ± 1.00		1.50	2.08 ± 0.33	
noB	20	1.25	1.65 ± 0.39	●●●	0.00	4.88 ± 2.05	○○○	5.99	5.51 ± 0.77	●●●
	10	8.75	8.72 ± 1.09	●●●	0.00	12.30 ± 3.11	●○○	13.48	15.25 ± 1.03	●●●
	5	27.50	28.59 ± 1.49	●●●	44.44	39.61 ± 3.28	●●●	25.13	26.55 ± 0.69	●●●
	2	48.12	47.49 ± 0.71	●●●	61.98	61.51 ± 0.43	●●●	38.45	38.02 ± 0.51	●●●
	1	51.88	52.35 ± 0.57	●●●	64.76	64.17 ± 0.27	●●●	41.12	40.93 ± 0.54	●●●
LBQ_{rand}	20	0.00	0.50 ± 0.26	○○●	0.00	2.83 ± 1.64	○○○	3.00	2.56 ± 0.38	○○○
	10	0.00	0.45 ± 0.19	○○⋆	0.00	7.28 ± 2.32	●○○	4.49	4.71 ± 0.55	●○●
	5	5.00	5.22 ± 0.75	●○●	0.00	9.67 ± 2.55	●○⋆	8.99	8.72 ± 0.66	●○●
	2	21.88	22.83 ± 1.25	●○●	21.88	21.71 ± 3.39	●○●	21.80	21.48 ± 0.87	●○●
	1	44.38	44.15 ± 1.04	●○●	51.39	50.17 ± 1.79	●○●	36.78	36.38 ± 0.57	●○●
LBQ_{umda}	20	0.00	0.00 ± 0.00	●●⋆	0.00	2.44 ± 1.35	○○⋆	1.50	1.89 ± 0.45	○○○
	10	0.00	0.60 ± 0.24	○○○	0.00	9.94 ± 2.81	●○○	4.49	3.93 ± 0.45	●○●
	5	1.88	2.82 ± 0.57	●●⋆	0.00	10.44 ± 2.36	●○○	7.49	7.25 ± 0.57	●○●
	2	17.50	15.55 ± 1.02	●●●	0.00	6.77 ± 2.27	●●⋆	21.14	20.56 ± 0.83	●○●
	1	38.75	38.76 ± 1.22	●●●	35.07	34.01 ± 3.20	●●●	37.80	37.28 ± 0.88	●○●
$\text{LBQ}_{\text{comit}}$	20	0.00	0.10 ± 0.07	●○○	0.00	3.28 ± 1.54	○○○	1.50	1.66 ± 0.37	○○⋆
	10	0.00	0.70 ± 0.28	○○○	0.00	5.83 ± 2.17	●○⋆	3.00	2.47 ± 0.40	○●⋆
	5	3.75	3.60 ± 0.62	●○○	0.00	12.28 ± 2.80	●○○	5.99	5.20 ± 0.52	●●⋆
	2	10.62	10.88 ± 0.82	●●⋆	19.44	17.89 ± 3.08	●○●	14.98	14.23 ± 0.83	●●⋆
	1	27.50	26.92 ± 1.42	●●⋆	21.53	18.95 ± 3.54	●●⋆	32.14	32.88 ± 0.75	●●⋆

Tabla 5.7: Resultados del test de Holm ($\alpha = 0{,}05$) usando $\text{LBQ}_{\text{comit}}$ como algoritmo de control.

i	algoritmo	z-statistic	p-valor	α/i
1	LBQ_{umda}	5.657e–01	2.858e–01	5.000e–02
2	LBQ_{rand}	1.980e+00	2.386e–02	2.500e–02
3	noB	5.091e+00	1.779e–07	1.667e–02

LBQ_{rand}, exhibiendo una pendiente más baja, es decir, una degradación menor conforme se incrementa la tasa de *churn*. Esto se puede ver más claramente en la Figura 5.9(b) en la cual se muestra una gráfica con la distribución de los rankings relativos (1 para el mejor, 4 para el peor) de cada algoritmo sobre cada problema y escenario de *churn*.

La ventaja de las variantes de LBQ sobre noB así como la ventaja de $\text{LBQ}_{[\text{umda}|\text{comit}]}$ sobre LBQ_{rand} es clara. No obstante, para confirmar si estas diferencias son significativas se ha llevado a cabo el test de Quade, obteniendo que al menos un algoritmo tiene diferencias significativas sobre el resto (p-valor



(a) LBQ_{umda} (b) $\text{LBQ}_{\text{comit}}$

Figura 5.10: Evolución del mejor fitness en TRAP para LBQ_{umda} y $\text{LBQ}_{\text{comit}}$ en función de la tasa de *churn*.

$\simeq 0$). Posteriormente se ha realizado el test post-hoc de Holm (ver Tabla 5.7) usando $\text{LBQ}_{\text{comit}}$ como algoritmo de control. El resultado muestra que hay diferencias significativas tanto con noB como con LBQ_{rand} en el nivel $\alpha = 0{,}05$. La diferencia entre LBQ_{umda} y $\text{LBQ}_{\text{comit}}$ no es estadísticamente significativa en este nivel en una escala global. Una perspectiva más fina se muestra en la Tabla 5.6. En general $\text{LBQ}_{\text{comit}}$ proporciona los mejores resultados y es mejor (con diferencias estadísticamente significativas) que LBQ_{umda} en la mayoría de escenarios de *churn* extremos. Esto se ilustra más en profundidad en la Figura 5.10 en la cual se muestra la evolución del mejor fitness (para la función TRAP) para las distintas variantes del modelo probabilístico utilizadas. El rendimiento es estadísticamente indistinguible para tasas de *churn* bajas o moderadas (superiores o iguales a $k = 5$) pero hay una clara superioridad (estadísticamente significativa) de $\text{LBQ}_{\text{comit}}$ en los dos escenarios más severos. Esto se puede explicar por los siguientes motivos:

1. El agrandamiento de la población se demanda con menos frecuencia en escenarios con una tasa de *churn* baja, y por lo tanto la estrategia concreta elegida afecta en menor medida al rendimiento, ya que se ejecutará un número menor de veces.

2. La mejor precisión del modelo bivariable para representar la población parece que es decisiva para mantener el progreso de la búsqueda en escenarios con tasas de *churn* extremadamente altas.

5.3.3. Discusión

Las estrategias de auto-muestreo intentan producir nuevas soluciones (utilizadas para agrandar las poblaciones de una isla cuando sea necesario) similares pero no idénticas a otras soluciones de una población determinada para que la distorsión que pueda causar el *churn* sobre el proceso de búsqueda se reduzca al mínimo. Se ha mostrado que el uso de estas estrategias puede mejorar el rendimiento del algoritmo base, manteniendo mejor el impulso de la búsqueda, sobre todo en situaciones con tasas de *churn* extremadamente altas. Se han considerado dos aproximaciones diferentes para construir el modelo probabilístico, basadas tanto en dependencias univariables como bivariables. Aunque ambas aproximaciones proporcionan globalmente resultados comparables, las últimas parecen superiores cuando la volatilidad del sistema es elevada, lo que se puede interpretar como que es necesario utilizar modelos de población lo más precisos posible en tales escenarios altamente inestables.

5.4. Estrategias de Auto-Cableado

En esta sección se va a tratar una forma de auto-reparación (o auto-curación como se vio en la Sección 2.5.2.3) consistente en un auto-cableado dinámico de los nodos de la red para tratar el efecto de la pérdida de conectividad causado por la volatilidad del sistema, es decir, el cambio en tiempo real de los enlaces entre las islas para mantener el nivel de conexionado a pesar de las caídas de algunos nodos.

5.4.1. Auto-Cableado Dinámico

La red de interconexión utilizada es de escala libre, generada usando el modelo de Barabási-Albert (BA) como se vio en la Sección 4.2.2. Un ejemplo de aplicación de este modelo se muestra en la Figura 5.11 para $n_\iota = 32$ y $m = 2$. Como se puede ver, la conexión preferente causa que la red se caracterice por tener unos pocos nodos con muchas conexiones y cada vez más nodos con pocas conexiones, siguiendo el patrón bien conocido de conectividad de las redes de escala libre. Considérese ahora cómo evoluciona el sistema cuando los fallos en los nodos empiezan a tener lugar (Figura 5.12, fila superior): la red cada vez se vuelve más dispersa e incluso se desconecta, conteniendo subredes aisladas y nodos sueltos. Esto puede perjudicar el funcionamiento del algoritmo de diferentes formas: en primer lugar, esto limita el flujo de información a través de la migración entre las islas, algo fundamental en este modelo; en segundo lugar, perturba el funcionamiento del algoritmo de equilibrado provocando no solo la pérdida cuantitativa de individuos sino también más frecuentes reinicializaciones de islas desde cero (debido a la pérdida de eficiencia del equilibrado), dificultando el progreso de la búsqueda.

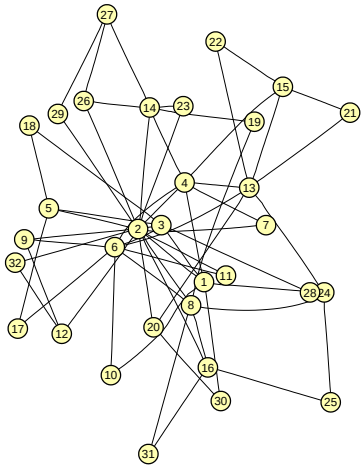
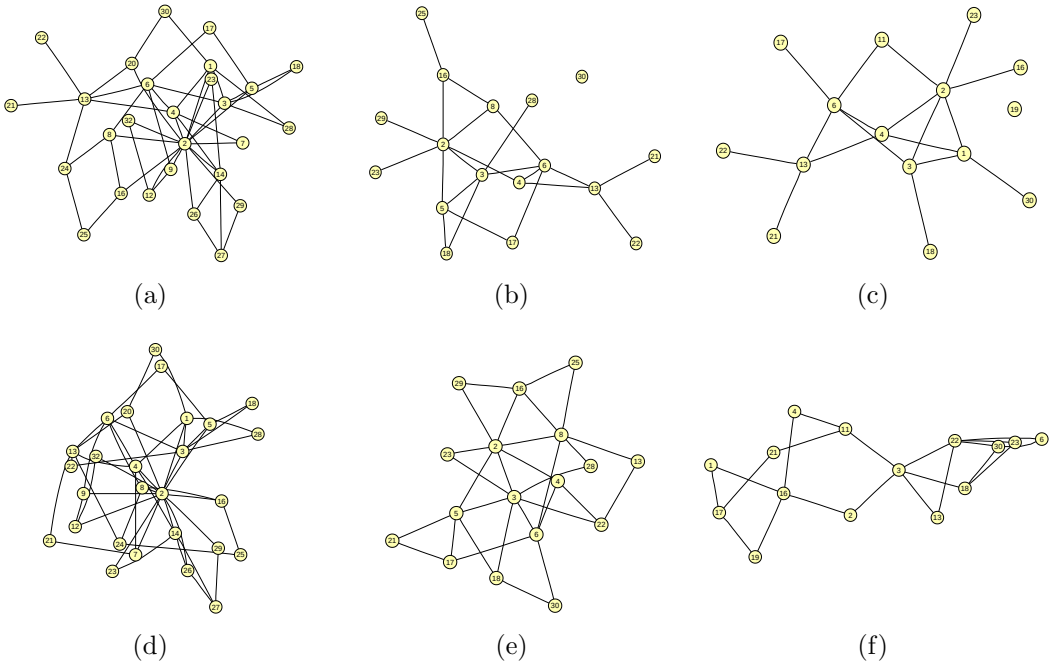


Figura 5.11: Ejemplo de red de escala-libre generada con el modelo de Barabási-Albert ($n_\iota = 32$, $m = 2$).



(a) (b) (c)

(d) (e) (f)

Figura 5.12: Comparación de la evolución de la red volátil de la Figura 5.11 sin auto-cableado (a)-(c) y con auto-cableado (d)-(f). Se muestran tres instantáneas del estado de la red en $t = 100, 250, 500$ usando $n_\iota = 32$ islas y $k = 10$ (ver Sección 5.4.2).

Para mitigar estos problemas se considera el uso de estrategias de auto-cableado. Estas estrategias funcionan de la siguiente manera:

1. Los vecinos que están inactivos durante la etapa de equilibrado se olvidan.

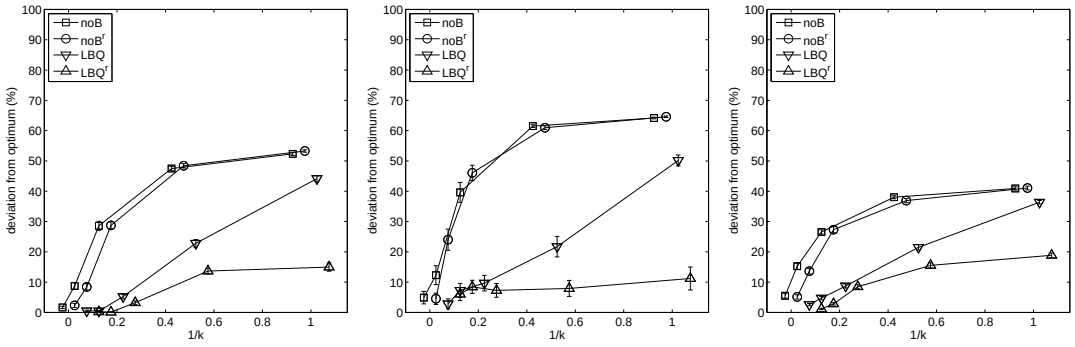
2. Cuando una isla detecta que el número de sus vecinos activos ha caído por debajo de un umbral predefinido (en este caso, el valor del parámetro m del modelo BA usado para crear la red), busca vecinos adicionales hasta alcanzar el nivel mínimo. Aunque esto se puede hacer de una forma eminentemente descentralizada usando el algoritmo de formación de tríadas [176] o el protocolo *newscast* [187], se ha considerado en este trabajo una alternativa más sencilla basada en el uso del propio modelo BA. Esto sirve como una demostración de concepto sobre la utilidad de la aproximación y allana el camino para usarlo eventualmente en otras aproximaciones de auto-cableado.

La fila inferior de la Figura 5.12 muestra el estado de la red en el mismo escenario de activación/desactivación ilustrado en la fila superior cuando se utiliza auto-cableado. Puede comprobarse como la red mantiene una elevada conectividad y se consigue evitar el aislamiento de los nodos.

5.4.2. Análisis Experimental

Se han considerado $n_\iota = 32$ islas con tamaño inicial de $\mu = 16$ individuos y un número total de 50000 evaluaciones. Las longitudes de los memes evolucionan entre $l_{\text{mín}} = 3$ y $l_{\text{máx}} = 9$, mutando sus longitudes con una probabilidad $p_r = 1/9$. Se utiliza una probabilidad de cruce $p_X = 1{,}0$ y una probabilidad de mutación $p_M = 1/\ell$, donde ℓ es la longitud del genotipo. El parámetro m del modelo de Barabási-Albert es $m = 2$ y la probabilidad de migración es $p_{mig} = 1/80$. Con respecto a la desactivación/reactivación de los nodos, se ha utilizado el parámetro de forma $\eta = 1{,}5$ para tener una tasa de riesgo creciente, y los parámetros de escala $\beta = -1/\log(p)$ con $p = 1 - (kn_\iota)^{-1}$, $k \in \{1, 2, 5, 10, 20, \infty\}$. El parámetro C_1 utilizado durante una eventual reinicialización de las islas desde cero tiene el valor $2\mu = 32$. Se han realizado 25 simulaciones para cada algoritmo y escenario de *churn*. Se denominan como noB y LBQ las variantes algorítmicas sin equilibrado y con equilibrado respectivamente. Además, se utiliza el superíndice r para hacer referencia al uso del auto-cableado (i.e., noB^r y LBQ^r). Se han considerado tres funciones de test: TRAP (concatenando 32 trampas de 4 bits), H-IFF (usando 128 bits) y MMDP (usando 24 bloques de seis bits) –ver Apéndice A.

La Figura 5.13 muestra los resultados obtenidos en términos de la desviación con respecto a la solución óptima para cada uno de los problemas y para cada uno de los algoritmos en función de la tasa de *churn* (los datos numéricos correspondientes se proporcionan en la Tabla 5.8). Es evidente que el rendimiento se degrada conforme se incrementa la tasa de *churn*, pero la tendencia de la degradación es bastante diferente para los distintos algoritmos. Considérese en primer lugar que las variantes que usan equilibrado tienen un rendimiento notablemente mejor que sus homólogas sin equilibrado. Esto indica que el equilibrado está contribuyendo efectivamente al mantenimiento del impulso de la búsqueda a pesar de la volatilidad del sistema. Sin embargo, se observa como conforme se incre-



(a) (b) (c)

Figura 5.13: Desviación con respecto a la solución óptima en función de la tasa de *churn*. De izquierda a derecha: TRAP, H-IFF y MMDP.

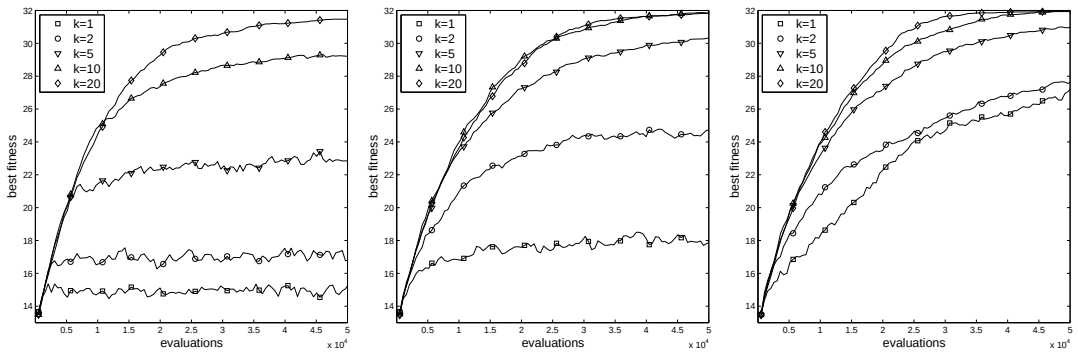
menta la tasa de *churn* (la zona de más a la derecha de las figuras) el rendimiento de LBQ se deteriora a un ritmo rápido, acercándose al pobre rendimiento de las variantes sin equilibrado (noB). Esta es una señal clara de que la inestabilidad creciente de la red subyacente está invalidando la efectividad de la estrategia de equilibrado. De hecho, LBQ^r tiene una curva de degradación mucho más suave, siendo considerablemente superior al resto de algoritmos sobre escenarios con elevadas tasas de *churn*. Esta superioridad se puede validar a través de un test signrank en el nivel $\alpha = 0{,}05$. Obsérvese también que el uso del auto-cableado no tiene efectos sobre el rendimiento de noB, cuyo comportamiento es virtualmente indistinguible de noB^r. Esto confirma que ninguna de las dos estrategias, tanto el auto-equilibrado de carga como el auto-cableado, es capaz por sí misma de asegurar la fortaleza del rendimiento del algoritmo (aunque es cierto que el equilibrado tiene un impacto mayor cuando las tasas de *churn* son más bajas). Por el contrario, se observa la sinergia entre las dos estrategias en el comportamiento de LBQ^r. Esto último se puede ver en la Figura 5.14 en la cual la evolución del fitness sobre la función TRAP se muestra para diferentes tasas de *churn*: obsérvese como el uso del equilibrado obtiene resultados análogos a las versiones sin equilibrado en los escenarios más estables, pero la búsqueda no es capaz de progresar mucho en los escenarios con tasas de *churn* más elevadas; en cambio el auto-cableado permite vencer esta situación, favoreciendo el progreso prolongado de la búsqueda incluso en estos últimos escenarios.

Una visión más profunda sobre el efecto comparado del auto-cableado se puede obtener a través del análisis espectral de la dinámica del tamaño de las islas. Se ha calculado la Densidad Espectral de Potencia (PSD)[2] de la evolución del tamaño de las islas en cada ejecución de LBQ y LBQ^r y se ha estimado la relación entre la frecuencia y la PSD a través de una ley de potencias PSD $\sim f^{\gamma}$. La Figura 5.15(a) muestra los valores de las pendientes del espectro obtenidas (γ). Para tasas bajas de *churn*, γ está próximo a -2, lo que indica ruido de Brown.

[2] *Power spectral density*

Tabla 5.8: Resultados (promediados para 25 ejecuciones) de los diferentes MMAs sobre los tres problemas considerados. Se muestra la mediana ($\tilde{x}$), la media ($\bar{x}$) y el error estándar de la media($\sigma_{\bar{x}}$).

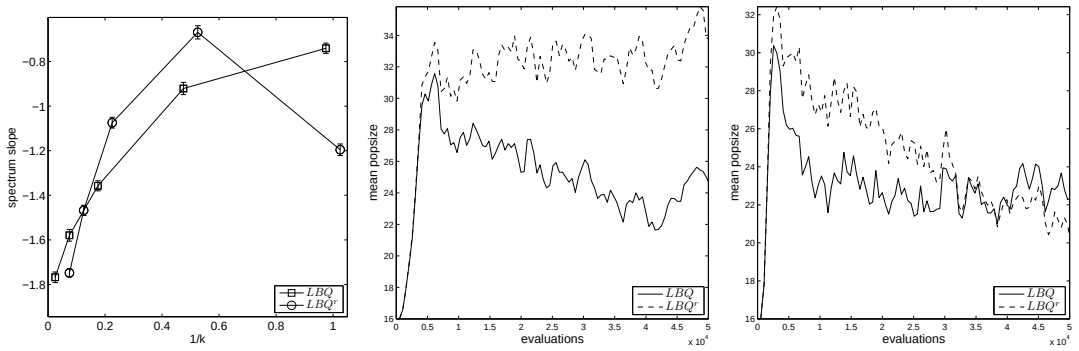
		TRAP		H-IFF		MMDP	
estrategia	k	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$
–	∞	0.00	0.55 ± 0.18	0.00	1.00 ± 1.00	1.50	2.08 ± 0.33
noB	20	1.25	1.65 ± 0.39	0.00	4.88 ± 2.05	5.99	5.51 ± 0.77
	10	8.75	8.72 ± 1.09	0.00	12.30 ± 3.11	13.48	15.25 ± 1.03
	5	27.50	28.59 ± 1.49	44.44	39.61 ± 3.28	25.13	26.55 ± 0.69
	2	48.12	47.49 ± 0.71	61.98	61.51 ± 0.43	38.45	38.02 ± 0.51
	1	51.88	52.35 ± 0.57	64.76	64.17 ± 0.27	41.12	40.93 ± 0.54
noB^r	20	1.25	2.32 ± 0.65	0.00	4.50 ± 1.88	4.49	5.11 ± 0.71
	10	9.38	8.43 ± 1.06	24.65	24.01 ± 3.49	13.48	13.62 ± 0.93
	5	29.37	28.73 ± 1.14	50.52	46.08 ± 2.51	28.30	27.30 ± 1.03
	2	48.75	48.40 ± 0.73	61.63	60.96 ± 0.64	37.29	36.91 ± 0.67
	1	53.13	53.28 ± 0.40	64.76	64.57 ± 0.25	41.12	41.05 ± 0.62
LBQ	20	0.00	0.50 ± 0.26	0.00	2.83 ± 1.64	3.00	2.56 ± 0.38
	10	0.00	0.45 ± 0.19	0.00	7.28 ± 2.32	4.49	4.71 ± 0.55
	5	5.00	5.22 ± 0.75	0.00	9.67 ± 2.55	8.99	8.72 ± 0.66
	2	21.88	22.83 ± 1.25	21.88	21.71 ± 3.39	21.80	21.48 ± 0.87
	1	44.38	44.15 ± 1.04	51.39	50.17 ± 1.79	36.78	36.38 ± 0.57
LBQ^r	20	0.00	0.20 ± 0.16	0.00	6.06 ± 2.14	1.50	1.14 ± 0.22
	10	0.00	0.10 ± 0.07	0.00	8.44 ± 2.17	3.00	2.82 ± 0.40
	5	1.88	3.20 ± 0.60	0.00	7.27 ± 2.28	8.66	8.48 ± 0.65
	2	12.50	13.65 ± 0.90	0.00	7.92 ± 2.63	14.65	15.53 ± 0.87
	1	16.25	14.97 ± 1.36	0.00	11.20 ± 3.79	19.47	18.86 ± 0.73



(a) (b) (c)

Figura 5.14: Evolución del mejor fitness para la función TRAP para diferentes tasas de *churn*. (a) noB (b) LBQ (c) LBQ^r. Los resultados para noB^r son muy similares a noB.

Esto se puede interpretar como consecuencia de la baja volatilidad de los nodos, que permite que el algoritmo tenga tiempo para equilibrar los tamaños de las islas; la desactivación/reactivación de las islas provoca que el tamaño medio de las islas siga una trayectoria browniana. Conforme la tasa de *churn* se incrementa, los cambios en la disponibilidad de los nodos interfieren con la operación de equilibrado; el sistema no es capaz de llegar a un estado estable entre los even-



(a) (b) (c)

Figura 5.15: (a) Pendiente de la PSD de la evolución del tamaño medio de las islas. (b) Tamaño medio de islas con/sin auto-cableado para LBQ con $k = 5$. (c) Idem con $k = 2$.

tos de desactivación/reactivación provocando nuevos flujos de equilibrado que se solapan con los anteriores. Como resultado la dinámica del tamaño de las islas empieza a moverse a un régimen próximo al ruido rosa ($\gamma = -1$), lo que es un signo de auto-organización del sistema –ver Figuras 5.15(b) y 5.15(c) para una representación de las trayectorias del tamaño medio de las islas para $k = 5$ (la pendiente de la PSD de LBQ^r está próxima a -1) y $k = 2$ (idem para LBQ) respectivamente. Se puede conjeturar que las diferencias entre LBQ y LBQ^r para las tasas de *churn* más altas se deben al hecho de que el proceso de equilibrado se ve seriamente perjudicado en LBQ, causando muchas inicializaciones de islas desde cero, mientras que el equilibrado funciona en el caso de LBQ^r (no obstante fallos simultáneos en los nodos provocan que la red decremente el tamaño total de su población –cf. Sección 5.1.2; obsérvese que en cualquier caso LBQ^r es lo bastante resistente para salir con éxito de estos decrementos como se muestra en la Figura 5.13).

5.4.3. Discusión

Las estrategias de auto-equilibrado de carga junto con una política de auto-cableado dinámico en MMAs basados en islas que se ejecutan en entornos de computación inestables son una opción interesante para desarrollar EAs robustos y tolerantes a la volatilidad del sistema. En este sentido, se ha mostrado que el uso aislado de estrategias de equilibrado no es suficiente para mitigar la degradación del rendimiento en escenarios con tasas de *churn* extremadamente elevadas. Esto origina la necesidad de complementar estas estrategias con otras como las políticas de auto-cableado que son capaces de mantener el patrón de conectividad en la red global, evitando la desconexión de partes de la misma o la aparición de cuellos de botella que perturben la efectividad del flujo de información a lo largo de la red.

5.5. Auto-Cableado con Auto-Muestreo

En las secciones anteriores de este capítulo se ha visto como las estrategias de auto-equilibrado de la carga de los nodos, la topología subyacente, el auto-muestreo de poblaciones o el auto-cableado dinámico de la red inciden en el rendimiento final del algoritmo. En esta sección se combinan todas ellas, dando lugar a un modelo de MMA con múltiples capacidades de auto-adaptación.

5.5.1. Análisis Experimental

Los experimentos se han realizado usando un MMA con $n_\iota = 32$ islas cuyo tamaño inicial es $\mu = 16$ individuos interconectados utilizando una topología de escala libre generada con el modelo de Barabási-Albert usando el parámetro $m = 2$. Se considera una probabilidad de cruce $p_X = 1{,}0$, una probabilidad de mutación $p_M = 1/\ell$ –donde ℓ es la longitud del genotipo– y una probabilidad de migración $p_{\text{mig}} = 1/80$. Los memes tienen longitudes que varían desde $l_{\text{mín}} = 3$ hasta $l_{\text{máx}} = 9$ y se muta su longitud con probabilidad $p_r = 1/l_{\text{máx}}$. Para controlar el *churn*, se considera que la distribución de Weibull que describe la disponibilidad de los nodos se define con el parámetro de forma $\eta = 1{,}5$ y por seis parámetros de escala $\beta = -1/\log(p)$ para $p = 1 - (kn_\iota)^{-1}$, con $k \in \{1, 2, 5, 10, 20, \infty\}$. El parámetro C_1 utilizado durante una eventual reinicialización de las islas desde cero se ha fijado al valor $2\mu = 32$. Se han realizado 25 simulaciones para cada algoritmo y escenario de *churn*, cada una de ellas constando de 50000 evaluaciones. Los diferentes MMAs se denominan LBQ_{rand}, LBQ_{umda} y $\text{LBQ}_{\text{comit}}$ según usen reinicialización aleatoria de los individuos o modelos probabilísticos univariables (UMDA) o bivariables (COMIT) respectivamente. Además, se utiliza el superíndice r para hacer referencia al uso del auto-cableado. Para poder establecer comparaciones que sirvan como referencia se ha considerado también un MMA sin equilibrado de carga que se ha denominado como noB. Se han considerado tres funciones de test: TRAP (concatenando 32 trampas de cuatro bits cada una), H-IFF (usando 128 bits) y MMDP (usando 24 bloques de seis bits cada uno) –ver el Apéndice A para más detalles.

En primer lugar se observa como es habitual que la degradación del rendimiento aumenta conforme se incrementa la tasa de *churn*, lo que es coherente con los resultados obtenidos para los modelos que utilizan tan solo una parte de las características de este. Por otro lado se pueden observar algunas particularidades haciendo un análisis más detallado a través de las dos dimensiones diferentes objeto de esta sección: el auto-cableado dinámico y el auto-muestreo. En la Figura 5.16 se agrupan las curvas en función de la versión algorítmica utilizada: versión cualitativa o cuantitativa junto con el auto-cableado dinámico activo o inactivo y en la Figura 5.17 en función del modelo de reinicialización empleado: aleatorio, por muestreo univariable (UMDA) o por muestreo bivariable (COMIT). Con respecto a lo primero, se puede comprobar que las versiones con auto-equilibrado cuantitativo tienen un comportamiento prácticamente idéntico,

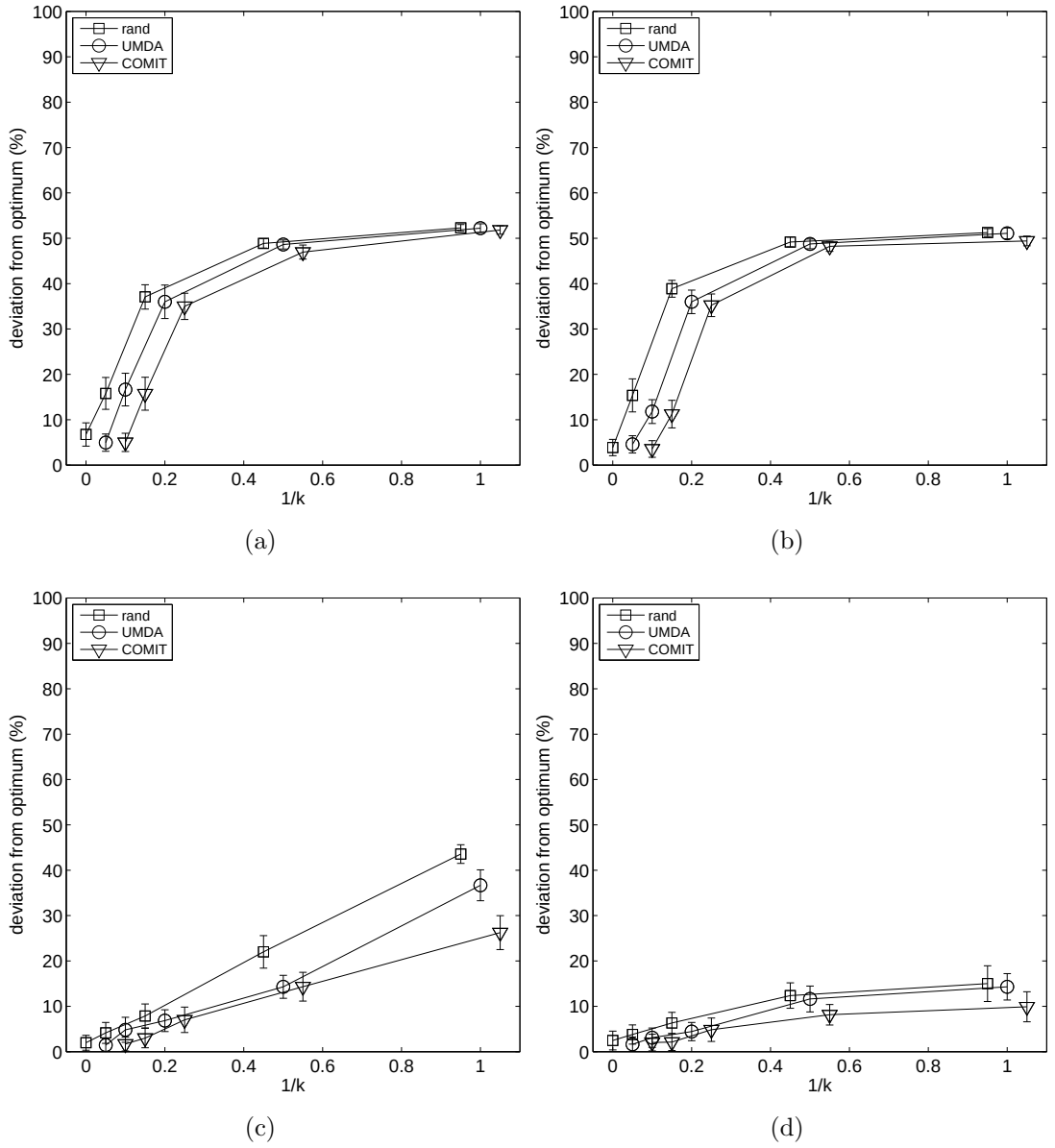


Figura 5.16: Desviación con respecto a la solución óptima en función de la tasa de *churn*. Las curvas se han agrupado por versión algorítmica: (a) LB, (b) LB^r, (c) LBQ y (d) LBQ^r.

tanto si se utiliza reinicialización aleatoria como por muestreo probabilístico, con o sin auto-cableado dinámico de los nodos, lo que seguramente está indicando que *LB* es una estrategia que es incapaz de mitigar las pérdidas de rendimiento conforme se incrementa la tasa de *churn*, independientemente del resto de estrategias utilizadas para gestionar la volatilidad del sistema –ver gráficas de las Figuras 5.16 (a) y (b). El escenario es considerablemente diferente para *LBQ*. En

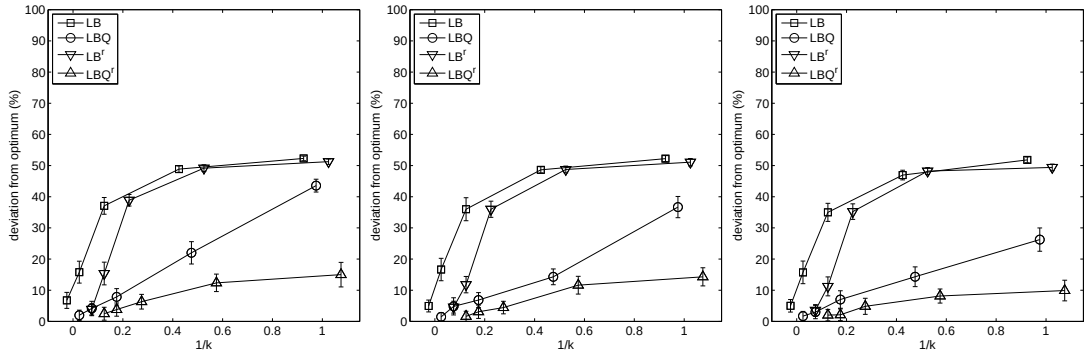


Figura 5.17: Desviación con respecto a la solución óptima en función de la tasa de *churn*. Las curvas se han agrupado por modelo de reactivación: (a) aleatoria, (b) UMDA y (c) COMIT.

este caso hay dos cuestiones interesantes que hay que mencionar. Por un lado, el rendimiento es mejor en las versiones con auto-muestreo que en la versión con reinicialización aleatoria (mejor con COMIT que con UMDA) y por otro, cuando se utiliza el auto-cableado dinámico se obtiene una menor degradación del rendimiento. Esto puede observarse en las Figuras 5.16(c) y(d). Estos resultados se corroboran a través del análisis por la dimensión del tipo de reinicialización utilizada (ver Figura 5.17 donde se observa que: (i) las curvas de LB y LB^r son prácticamente iguales para los tres tipos de reinicialización, con lo que se refuerza la idea de que la variante cuantitativa del auto-equilibrado es incapaz de mejorar el rendimiento del algoritmo aunque se complemente esta estrategia con otras y (ii) en la versión con auto-equilibrado cualitativo, cuando además se incluye el auto-cableado dinámico, mejora notablemente el rendimiento en los tres tipos de reinicialización. En las Tablas 5.9 y 5.10 se muestran los resultados completos detallados.

Finalmente, en la Figura 5.18 se muestran los resultados de la evolución del mejor fitness para un problema concreto (MMDP en este caso, aunque se podrían extrapolar al resto de problemas considerados). En ellas se puede comprobar como para tasas de *churn* bajas, apenas hay diferencias entre las versiones con/sin auto-cableado, en cambio, en los escenarios más volátiles (k=1 y k=2) las curvas de LBQ –para todos los modelos de reinicialización– tienen una pendiente considerablemente mayor cuando el algoritmo se ha ejecutado con auto-cableado (columna derecha de la Figura 5.18). Esto afianza lo expuesto en el párrafo precedente y se puede justificar a partir del hecho de que la pérdida de conexiones entre los nodos incide negativamente en los procesos de migración entre islas y en el propio auto-equilibrado, perjudicando tanto la diversidad genética como memética del algoritmo e impidiendo que este mejore.

Tabla 5.9: Resultados (promediados para 25 ejecuciones) de los diferentes MMAs sobre los tres problemas considerados sin auto-cableado. Se muestra el número de veces que se alcanza el óptimo (n_{opt}), la mediana ($\tilde{x}$), la media ($\bar{x}$) y el error estándar de la media ($\sigma_{\bar{x}}$). Los tres símbolos próximos a cada entrada indican si las diferencias son estadísticamente significativas (•) o no lo son (○). El primer símbolo corresponde a una comparación entre el algoritmo correspondiente y la versión sin fallos ($k = \infty$); el segundo refleja una comparación con LBQ_{rand} y el tercero es una comparación con el algoritmo que proporciona el mejor resultado para el correspondiente problema y valor de k de entre todas las versiones, con y sin reconexionado (identificado mediante ⋆).

		TRAP				H-IFF				MMDP			
estrategia	k	n_{opt}	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$		n_{opt}	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$		n_{opt}	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$	
–	∞	17	0.00	0.55 ± 0.18		24	0.00	1.00 ± 1.00		5	1.50	2.08 ± 0.33	
noB	20	12	1.25	1.65 ± 0.39	●○●	20	0.00	4.88 ± 2.05	○○○	3	5.99	5.51 ± 0.77	●○●
	10	1	8.75	8.72 ± 1.09	●○●	14	0.00	12.30 ± 3.11	●●●	0	13.48	15.25 ± 1.03	●○●
	5	0	27.50	28.59 ± 1.49	●○●	2	44.44	39.61 ± 3.28	●○●	0	25.13	26.55 ± 0.69	●○●
	2	0	48.12	47.49 ± 0.71	●○●	0	61.98	61.51 ± 0.43	●○●	0	38.45	38.02 ± 0.51	●○●
	1	0	51.88	52.35 ± 0.57	●○●	0	64.76	64.17 ± 0.27	●○●	0	41.12	40.93 ± 0.54	●○●
LB_{rand}	20	6	5.00	4.75 ± 0.80	●●●	19	0.00	6.42 ± 2.45	●○○	0	8.99	9.07 ± 0.67	●●●
	10	0	12.50	14.48 ± 1.48	●○●	11	19.44	15.56 ± 3.17	●○●	0	17.64	17.34 ± 1.06	●○●
	5	0	33.75	32.79 ± 1.67	●○●	0	51.54	49.00 ± 2.15	●○●	0	30.46	29.43 ± 0.57	●○●
	2	0	46.88	47.02 ± 0.77	●○●	0	62.33	62.02 ± 0.41	●○●	0	37.29	37.57 ± 0.63	●○●
	1	0	53.08	52.08 ± 0.55	●○●	0	64.58	64.55 ± 0.19	●○●	0	40.79	40.25 ± 0.50	●○●
LBQ_{rand}	20	21	0.00	0.50 ± 0.26	○○●	22	0.00	2.83 ± 1.64	○○○	7	3.00	2.56 ± 0.38	○●●
	10	19	0.00	0.45 ± 0.19	○○○	17	0.00	7.28 ± 2.32	●○○	1	4.49	4.71 ± 0.55	●●●
	5	3	5.00	5.22 ± 0.75	●○●	15	0.00	9.67 ± 2.55	●○○	0	8.99	8.72 ± 0.66	●○●
	2	0	21.88	22.83 ± 1.25	●●●	7	21.88	21.71 ± 3.39	●●●	0	21.80	21.48 ± 0.87	●●●
	1	0	44.38	44.15 ± 1.04	●●●	0	51.39	50.17 ± 1.79	●●●	0	36.78	36.38 ± 0.57	●●●
LB_{UMDA}	20	5	3.75	4.10 ± 0.89	●●●	22	0.00	2.65 ± 1.64	○○○	0	7.49	8.08 ± 0.71	●○●
	10	0	16.25	15.73 ± 1.06	●●●	11	16.67	16.22 ± 3.41	●○●	0	17.97	17.99 ± 1.08	●○●
	5	0	34.38	33.17 ± 1.64	●○●	0	51.48	45.74 ± 3.32	●○●	0	28.12	29.07 ± 0.98	●○●
	2	0	46.25	46.18 ± 0.71	●○●	0	61.98	62.01 ± 0.44	●○●	0	38.65	37.70 ± 0.61	●○●
	1	0	52.50	51.92 ± 0.76	●○●	0	64.06	63.70 ± 0.33	●○●	0	41.63	41.05 ± 0.54	●●●
LBQ_{UMDA}	20	25	0.00	0.00 ± 0.00	●●⋆	22	0.00	2.44 ± 1.35	○○⋆	8	1.50	1.89 ± 0.45	○○○
	10	19	0.00	0.60 ± 0.24	○○○	16	0.00	9.94 ± 2.81	●○○	2	4.49	3.93 ± 0.45	●●●
	5	9	1.88	2.82 ± 0.57	●●●	13	0.00	10.44 ± 2.36	●○○	0	7.49	7.25 ± 0.57	●○●
	2	0	17.50	15.55 ± 1.02	●●●	18	0.00	6.77 ± 2.27	●○⋆	0	21.14	20.56 ± 0.83	●●●
	1	0	38.75	38.76 ± 1.22	●●●	3	35.07	34.01 ± 3.20	●●●	0	37.80	37.28 ± 0.88	●●●
LB_{COMIT}	20	7	2.50	2.80 ± 0.58	●●●	20	0.00	4.39 ± 1.89	○○○	0	7.49	7.82 ± 0.71	●●●
	10	0	10.62	11.07 ± 1.17	●●●	9	21.88	19.63 ± 3.45	●○●	0	17.36	16.49 ± 1.01	●○●
	5	0	32.50	30.20 ± 1.49	●○●	0	48.09	45.22 ± 2.48	●○●	0	29.80	29.55 ± 0.87	●○●
	2	0	45.63	44.39 ± 1.10	●○●	0	61.28	59.98 ± 0.99	●○●	0	36.45	36.49 ± 0.65	●○●
	1	0	51.88	51.52 ± 0.68	●●●	0	63.84	63.70 ± 0.27	●●●	0	40.28	40.23 ± 0.62	●○●
LBQ_{COMIT}	20	23	0.00	0.10 ± 0.07	●○○	21	0.00	3.28 ± 1.54	○○○	11	1.50	1.66 ± 0.37	○○○
	10	18	0.00	0.70 ± 0.28	○○○	19	0.00	5.83 ± 2.17	●○○	5	3.00	2.47 ± 0.40	○○○
	5	5	3.75	3.60 ± 0.62	●●●	13	0.00	12.28 ± 2.80	●○○	1	5.99	5.20 ± 0.52	●○○
	2	0	10.62	10.88 ± 0.82	●●●	9	19.44	17.89 ± 3.08	●●●	0	14.98	14.23 ± 0.83	●●●
	1	0	27.50	26.92 ± 1.42	●●●	10	21.53	18.95 ± 3.54	●○○	0	32.14	32.88 ± 0.75	●●●

5.5.2. Discusión

La utilización de propiedades self-⋆ tiene consecuencias positivas sobre el rendimiento de un MMA cuando este se ejecuta en un entorno computacional inestable. Sin embargo, estas propiedades incluidas de forma independiente tienen algunas limitaciones que se solucionan conforme se van añadiendo varias de estas propiedades al algoritmo, construyendo un modelo más completo. En esta sección se ha podido verificar que aunque las estrategias de auto-equilibrado cualitativo permiten obtener mejoras considerables en el rendimiento del algoritmo, si la agi-

Tabla 5.10: Resultados (promediados para 25 ejecuciones) de los diferentes MMAs sobre los tres problemas considerados con auto-cableado. Se muestra el número de veces que se alcanza el óptimo (n_{opt}), la mediana ($\tilde{x}$), la media ($\bar{x}$) y el error estándar de la media ($\sigma_{\bar{x}}$). Los tres símbolos próximos a cada entrada indican si las diferencias son estadísticamente significativas (•) o no lo son (○). El primer símbolo corresponde a una comparación entre el algoritmo correspondiente y la versión sin fallos ($k = \infty$); el segundo refleja una comparación con LBQ_{rand} y el tercero es una comparación con el algoritmo que proporciona el mejor resultado para el correspondiente problema y valor de k de todas las versiones, con y sin reconexionado (identificado mediante ⋆).

		TRAP				H-IFF				MMDP			
estrategia	k	n_{opt}	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$		n_{opt}	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$		n_{opt}	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$	
–	∞	17	0.00	0.55 ± 0.18		24	0.00	1.00 ± 1.00		5	1.50	2.08 ± 0.33	
noB^r	20	10	1.25	2.32 ± 0.65	●○●	20	0.00	4.50 ± 1.88	○○○	2	4.49	5.11 ± 0.71	●○●
	10	4	9.38	8.43 ± 1.06	●○●	7	24.65	24.01 ± 3.49	●●●	0	13.48	13.62 ± 0.93	●○●
	5	0	29.37	28.73 ± 1.14	●○●	1	50.52	46.08 ± 2.51	●○●	0	28.30	27.30 ± 1.03	●○●
	2	0	48.75	48.40 ± 0.73	●○●	0	61.63	60.96 ± 0.64	●○●	0	37.29	36.91 ± 0.67	●○●
	1	0	53.13	53.28 ± 0.40	●○●	0	64.76	64.57 ± 0.25	●○●	0	41.12	41.05 ± 0.62	●○●
LB^r_{rand}	20	14	0.00	1.45 ± 0.42	○●●	21	0.00	3.54 ± 1.71	○○○	0	7.49	6.60 ± 0.63	●●●
	10	0	13.75	13.14 ± 0.85	●○●	13	0.00	15.89 ± 3.62	●○●	0	17.64	17.06 ± 0.72	●○●
	5	0	35.00	34.95 ± 0.81	●○●	0	53.30	51.00 ± 1.63	●○●	0	30.46	30.66 ± 0.62	●○●
	2	0	48.12	48.55 ± 0.84	●○●	0	62.67	62.01 ± 0.50	●○●	0	36.96	37.00 ± 0.53	●○●
	1	0	51.88	51.14 ± 0.63	●○●	0	64.24	63.65 ± 0.37	●○●	0	38.79	38.99 ± 0.55	●○●
LBQ^r_{rand}	20	23	0.00	0.20 ± 0.16	●○○	18	0.00	6.06 ± 2.14	●○○	10	1.50	1.14 ± 0.22	●●○
	10	23	0.00	0.10 ± 0.07	●○⋆	15	0.00	8.44 ± 2.17	●○○	3	3.00	2.82 ± 0.40	○●●
	5	5	1.88	3.20 ± 0.60	●○●	17	0.00	7.27 ± 2.28	●○○	0	8.66	8.48 ± 0.65	●○●
	2	0	12.50	13.65 ± 0.90	●●●	18	0.00	7.92 ± 2.63	●●○	0	14.65	15.53 ± 0.87	●●●
	1	0	16.25	14.97 ± 1.36	●●●	18	0.00	11.20 ± 3.79	●●⋆	0	19.47	18.86 ± 0.73	●●●
LB^r_{UMDA}	20	12	1.25	1.30 ± 0.33	○●●	18	0.00	5.58 ± 1.89	●○○	0	7.16	6.84 ± 0.49	●○●
	10	0	8.75	9.00 ± 0.80	●●●	14	0.00	10.65 ± 2.56	●○○	0	16.14	15.74 ± 0.53	●○●
	5	0	33.13	32.61 ± 1.12	●○●	1	45.67	44.60 ± 2.39	●○●	0	30.13	30.76 ± 0.53	●○●
	2	0	47.50	47.36 ± 0.88	●○●	0	60.59	61.08 ± 0.44	●○●	0	38.12	37.82 ± 0.56	●○●
	1	0	51.25	50.70 ± 0.86	●○●	0	63.63	63.30 ± 0.40	●○●	0	38.96	39.24 ± 0.80	●●●
LBQ^r_{UMDA}	20	25	0.00	0.00 ± 0.00	●●●	20	0.00	3.89 ± 1.68	○○○	13	0.00	0.96 ± 0.23	●○○
	10	22	0.00	0.30 ± 0.18	○○○	18	0.00	6.39 ± 2.19	●○○	2	3.00	2.56 ± 0.28	○●●
	5	15	0.00	1.17 ± 0.51	○●⋆	18	0.00	5.81 ± 1.93	●○⋆	1	5.99	6.30 ± 0.62	●○●
	2	0	8.75	9.22 ± 0.77	●●○	14	0.00	11.27 ± 2.77	●○○	0	14.65	14.35 ± 0.73	●●●
	1	0	14.38	14.27 ± 0.91	●●●	8	20.83	17.19 ± 2.74	●●○	0	11.65	11.42 ± 0.89	●●●
LB^r_{COMIT}	20	16	0.00	0.90 ± 0.28	○●●	21	0.00	3.81 ± 1.82	○○○	1	5.66	5.94 ± 0.51	●●●
	10	2	6.87	7.00 ± 0.82	●●●	13	0.00	12.77 ± 2.95	●○●	0	13.48	13.94 ± 0.79	●○●
	5	0	31.25	30.54 ± 1.50	●○●	0	46.70	45.71 ± 2.01	●○●	0	29.80	29.44 ± 0.66	●○●
	2	0	45.63	45.64 ± 0.79	●○●	0	61.63	61.05 ± 0.52	●○●	0	38.48	37.99 ± 0.66	●○●
	1	0	47.50	47.91 ± 0.75	●●●	0	62.67	61.85 ± 0.58	●●●	0	39.29	38.46 ± 0.57	●○●
LBQ^r_{COMIT}	20	23	0.00	0.15 ± 0.11	●○○	19	0.00	5.06 ± 1.89	○○○	15	0.00	0.84 ± 0.25	●○⋆
	10	22	0.00	0.15 ± 0.08	○○○	20	0.00	4.61 ± 1.94	○○⋆	10	1.50	1.62 ± 0.31	○○⋆
	5	15	0.00	1.40 ± 0.48	○●○	16	0.00	8.94 ± 2.58	●○○	3	3.00	4.17 ± 0.57	●○⋆
	2	1	6.25	7.57 ± 0.76	●●⋆	17	0.00	6.97 ± 2.09	●●○	0	10.48	9.89 ± 0.76	●●⋆
	1	2	7.50	7.02 ± 0.82	●●⋆	10	20.83	18.52 ± 3.30	●○○	3	4.49	4.18 ± 0.59	●●⋆

tación del sistema es lo suficientemente elevada es necesario complementarlas con otras como el auto-muestreo o el auto-cableado (auto-cableado dinámico).

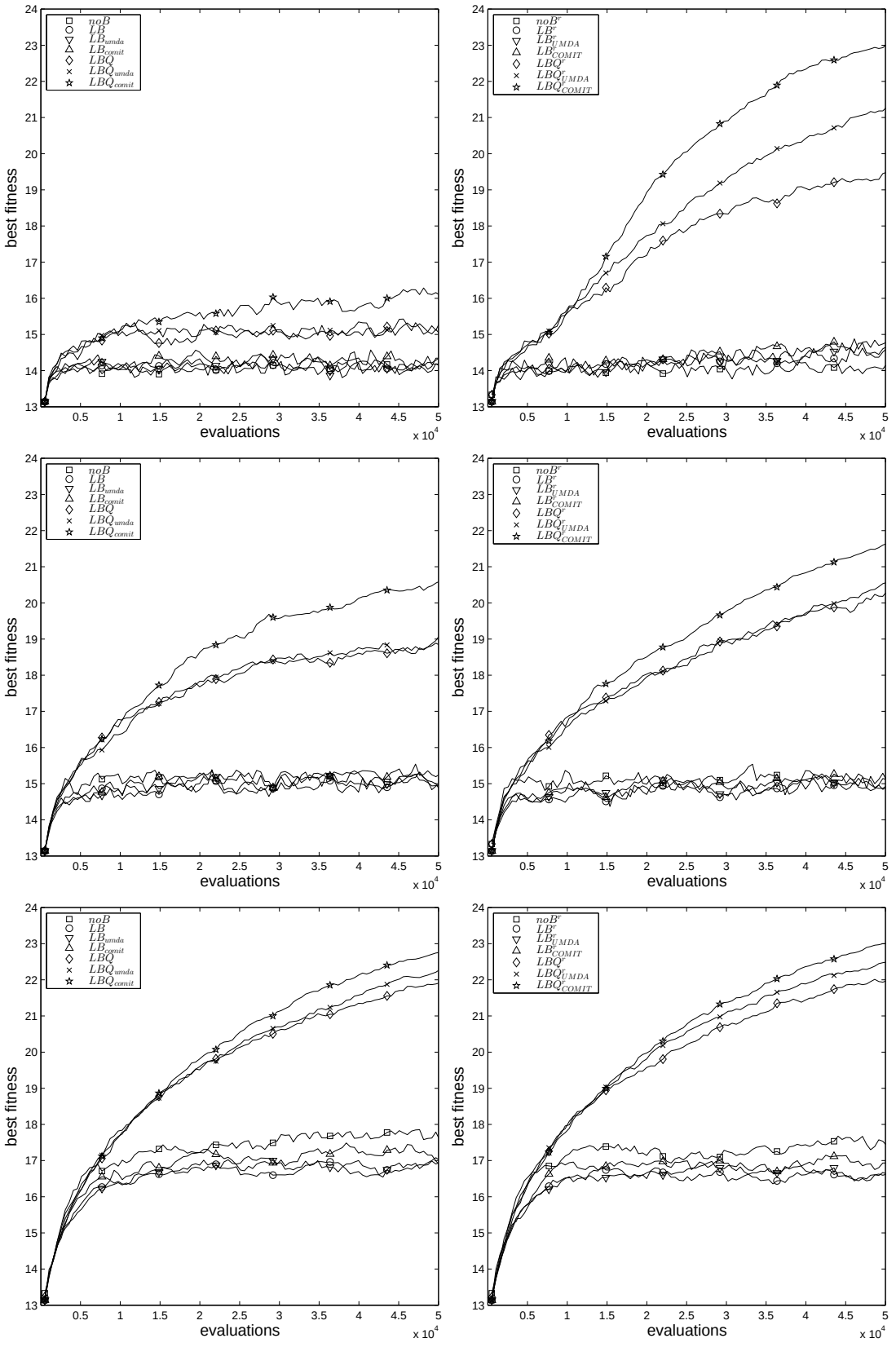


Figura 5.18: Evolución del mejor fitness para el problema MMDP. (Izquierda) Sin auto-cableado. (Derecha) Con auto-cableado. (De arriba a abajo) $k = 1, k = 2, k = 5$.

Capítulo 6

Discusión Final

En esta tesis se ha abordado el diseño y desarrollo de algoritmos meméticos con propiedades self-$\star$. Se ha considerado una clase de MMA con estructura espacial, dotando a la población de una cierta topología para restringir las interacciones entre los individuos y en la que los memes se definen como reglas de reescritura cuya longitud puede variar durante su ejecución, auto-adaptándose durante el proceso de búsqueda, estando sujetos a un proceso evolutivo similar al de la evolución genética. Se ha estudiado también la propagación y difusión de los memes a través de la población, lo que es considerablemente más complejo que cuando únicamente hay genes, ya que los efectos de los memes solo se pueden observar indirectamente por el efecto que producen sobre los genotipos.

A partir de este diseño principal se han añadido características adicionales al MMA, como la utilización de una aproximación alternativa al uso de los operadores genéticos tradicionales basada en EDAs, creando de esta forma un modelo probabilístico para la generación de la descendencia a partir de la población actual o los MMAs distribuidos basados en islas. En este último contexto, se ha analizado el papel que tienen las políticas de migración y el impacto que tienen los entornos inestables donde se ejecutan estos MMAs sobre el rendimiento de los algoritmos, lo que ha dado lugar a la necesidad de gestionar adecuadamente la volatilidad de las plataformas y el fenómeno del *churn* mediante la introducción de mecanismos de tolerancia a fallos (e.g. la creación de puntos de restauración periódicos) y la utilización de topologías complejas de interconexión de los nodos (redes SF y SW).

Una vez vistos algunos mecanismos correctivos para diseñar MMAs sobre entornos distribuidos inestables con elevadas tasas de volatilidad, se ha procedido a crear procedimientos para proporcionar al algoritmo la capacidad de detección de fallos con el objetivo de poder reaccionar ante el fenómeno del *churn*. Esto se ha conseguido mediante la introducción de técnicas de auto-equilibrado de la carga, de forma tal que el propio MMA sea capaz por sí mismo y sin necesidad de recurrir a un control central, de auto-adaptarse al *churn* variando los tamaños de sus poblaciones.

A pesar de todo lo anterior, sigue habiendo una degradación del rendimiento de

los algoritmos cuando la agitación del entorno es grande. Por ello y finalmente, se han incorporado procedimientos de auto-reparación para compensar el deterioro producido: (i) el auto-muestreo a través de un modelo probabilístico dinámico sobre las poblaciones de los nodos y (ii) el auto-cableado representado mediante un procedimiento de reconexión de los nodos en función de las desconexiones que se van produciendo en la red cuando un nodo abandona el sistema.

6.1. Conclusiones

El análisis de la propagación de memes en los MMAs ha mostrado que tanto la intensidad de la selección como las estructuras espaciales subyacentes juegan un papel muy importante en el hecho de que los memes de alto potencial tengan la oportunidad de proliferar en la población. Concretamente las estructuras espaciales (e.g. una rejilla cuadrada toroidal con topología de von Neumann) inducen tiempos de absorción más largos, lo que permite que los memes de mayor calidad incrementen sus tasas de supervivencia y por lo tanto tengan más oportunidades de expandirse mejorando los individuos que los contienen. Con respecto a la auto-adaptación memética, se ha corroborado que los MMA que disponen de este mecanismo (a través del cambio en las longitudes de los memes) consiguen mejorar los resultados de sus homólogos sin auto-adaptación y permiten ahorrar en tiempo de configuración del algoritmo, ya que el ajuste de los parámetros de los memes se puede sustituir por este proceso auto-adaptativo.

Por otra parte, la experimentación con EDAs multimeméticos ha proporcionado resultados notablemente buenos, sobre todo cuando se les dota de elitismo y se usan modelos probabilísticos de dos variables, ya que son claramente superiores a un MMA que manipule genes y memes mediante operadores genéticos.

Con respecto al modelo de islas se ha podido verificar que tanto la elección de los operadores de selección de migrantes como de la estrategia de reemplazo de migrantes tiene un impacto crucial en el rendimiento, aunque es mayor el de la selección. Algunas conclusiones concretas que se han podido extraer del análisis de las migraciones son: (i) una migración basada en el mejor individuo conduce a una rápida degradación de la diversidad y disminuye su rendimiento, (ii) una estrategia de selección puramente enfocada al mantenimiento de la diversidad del genotipo es inferior a las estrategias basadas en la diversidad memética y (iii) estas últimas tienen un funcionamiento estadísticamente similar a las estrategias de muestreo aleatorio de la población emisora.

La tolerancia a fallos y la resistencia al cambio son dos aspectos de importancia en los entornos emergentes para computación paralela, tales como las redes P2P o los entornos tipo *grid*. En este sentido se han considerado dos factores de diseño, por un lado la topología de interconexión entre los nodos y por otro la estrategia de gestión de fallos utilizada. Los experimentos que se han realizado han mostrado que las topologías estructuradas regularmente como las rejillas de von Neumann o los hipercubos parecen proporcionar un perfil más gradual de

degradación conforme se incrementa la tasa de *churn*. Desde el punto de vista de la política de gestión de fallos, la estrategia denominada checkpoint parece ser la elección más robusta, ya que es capaz de proporcionar un buen rendimiento en la mayoría de los escenarios, aunque los resultados también sugieren que cuando se utiliza una estrategia de reinicialización más simple junto con topologías de mayor rendimiento, tales como la rejilla de von Neumann o hipercubos, la degradación del sistema es comparable a la obtenida con la estrategia anterior. En esta línea, la estrategia checkpoint puede ser asequible en escenarios con tasas de *churn* bajas, sin embargo en escenarios caracterizados por tasas de *churn* elevadas donde se requieren backups frecuentes para poder hacer frente a la volatilidad de los nodos, se induce un sobrecoste adicional junto con la necesidad de tener acceso a dispositivos de almacenamiento externos persistentes, lo que hace que esta aproximación sea menos atractiva en tales situaciones.

El uso de estrategias de auto-equilibrado cualitativas (en la que la información genética y memética se transmite entre los nodos durante el proceso mismo de equilibrado de carga) cuyo objetivo es mantener el tamaño de la población global constante, tiene consecuencias positivas ante el fenómeno del *churn* y limita considerablemente la degradación del sistema frente a MMAs sin equilibrado. En relación a la inclusión de topologías de red complejas en MMAs y su efecto sobre el rendimiento, se ha observado que las redes de mundo pequeño (SW) proporcionan resultados notables para MMAs tolerantes a fallos con auto-equilibrado, aunque las redes de escala libre se han mostrado más eficientes, principalmente en los escenarios más volátiles. Otra cuestión interesante es que incrementando la densidad de la red el MMA parece ser más robusto, principalmente cuando el auto-equilibrado está también activo, lo que permite deducir que existen sinergias entre estos dos aspectos del algoritmo que hacen que tenga una mayor resistencia a las tasas de *churn* altas. Las estrategias de auto-muestreo probabilístico para producir nuevas soluciones (cuando se agrandan las poblaciones de las islas) también tienen un impacto positivo sobre el rendimiento de los MMAs basados en islas.

Finalmente, la utilización de estrategias de auto-equilibrado de carga para ajustar los tamaños de las poblaciones junto con el auto-cableado dinámico de los nodos de la red para tratar el efecto de la pérdida de conectividad (causado porque algunos nodos abandonen el sistema) es una opción sin duda interesante desde el punto de vista del rendimiento. Se ha mostrado que el uso de estrategias de equilibrado de manera aislada no es lo suficientemente robusto para minimizar la degradación del rendimiento en entornos altamente volátiles, lo que crea la necesidad de complementar esta estrategia con otras. En este sentido, el reconexionado dinámico de las poblaciones permite mantener el patrón de conectividad de la red global, evitando que se desconecten (y aíslen) partes de la misma.

En resumen, cuando se unen un número determinado de propiedades self-$\star$ se producen efectos beneficiosos para el rendimiento del algoritmo, considerablemente superiores al impacto que cada una de estas propiedades tiene por separado.

6.2. Trabajo Futuro

Las líneas de investigación que se pueden abrir a partir de este trabajo son múltiples. La más directa es la ampliación de todos los modelos vistos a MAs coevolutivos [330, 331, 334], en los cuales los memes se desacoplan de los genotipos y coevolucionan al mismo tiempo que ellos, pero en poblaciones independientes. Esta independencia entre los memes y los genes plantea nuevas decisiones de diseño, como elegir qué meme se aplica a qué genotipo o cómo se evalúan los memes, ya que al estar en una población diferente tienen que establecerse mecanismos propios de selección y reemplazo siguiendo algún criterio, seguramente definiendo una función de fitness específica para los memes, lo que puede ser en sí misma una nueva línea de investigación, ya que la medida de la calidad de los memes no es tan directa como en el caso de los genotipos. Una posibilidad es asignar a cada meme un valor (fitness) que determine la probabilidad de éxito del meme cuando se aplica a un individuo, lo que se podría calcular a partir del número de veces que la aplicación del meme produce un individuo de mejor calidad genotípica que el original.

Los memes también se pueden extender con reglas de reescritura más complejas, bien incorporando nuevos símbolos comodín en el consecuente que realicen operaciones unarias (inversiones [334]) o binarias sobre símbolos adyacentes o vecinos (en sentido general, i.e., a partir de una relación de vecindad definida sobre los componentes de un meme), o bien definidas sobre alfabetos no binarios utilizando hiperheurísticas que determinen de alguna manera la metaheurística que habría que aplicar en cada escenario, de esta forma se traslada el proceso de búsqueda desde un espacio de soluciones a un espacio de métodos de solución. Esto se podría utilizar cuando haya espacios combinatorios, e.g. permutaciones, y las reglas indicarían acciones como inversiones o intercambios entre posiciones de la permutación. Otra opción es definir memes transversales, es decir, cuya aplicación pueda afectar a más de un individuo a la vez, a diferencia de los utilizados en esta tesis, cuya aplicación únicamente altera el genotipo del individuo que lo contiene.

Con respecto a los modelos probabilísticos, se pueden desarrollar MMAs con modelos con dependencias multivariable (e.g. redes bayesianas) en alguna etapa del ciclo evolutivo o en la propia inicialización, o EDAs que sean capaces de capturar las dependencias multivariable y aplicarlos dentro de los mecanismos de tolerancia a fallos, como por ejemplo, en las reinicializaciones de los nodos o en el proceso migratorio de los individuos. Un buen ejemplo de EDA que se podría utilizar es el algoritmo BOA [297][1], basado precisamente en la construcción de una red bayesiana a partir de la población.

Otra posibilidad es utilizar nuevas topologías de interconexión entre los nodos y nuevos mecanismos de auto-cableado que sean capaces, bien de mantener la topología de la red o bien adaptarla al escenario de volatilidad existente en cada

[1] Bayesian Optimization Algorithm

momento, así como la incorporación de otras propiedades self-$\star$ al algoritmo en búsqueda de nuevas sinergias entre ellas que consoliden los resultados obtenidos o incluso los mejoren. Con respecto a lo primero, durante el propio proceso de auto-cableado se puede utilizar un parámetro m variable (número de enlaces por nodo) o utilizar un procedimiento diferente para reconstruir la red de forma que se mantenga su topología inicial. Para ello parecen interesantes las redes híbridas de escala libre/mundo pequeño [167], que son capaces de reproducir las distribuciones de los grados de las redes de escala libre y las propiedades de las redes de mundo pequeño. Sobre la exploración de nuevas propiedades self-$\star$, la auto-organización a través de estructuras de población dinámicas es una característica que puede ser estudiada con mayor profundidad [280].

Finalmente, hay que señalar que este trabajo ha tenido un fundamento eminentemente teórico o de investigación básica, aunque pensando siempre en una aplicación práctica eficiente sobre problemas del mundo real. Las aplicaciones de los MAs son amplias y variadas [256, 257, 261], por lo que una de las siguientes oportunidades y a la vez retos es conseguir aplicar los conceptos utilizados durante esta investigación en la resolución de problemas reales. El primer paso en este sentido consistiría en extender la representación de los MMAs a alfabetos arbitrarios. En esta línea, la escalabilidad es otra cuestión a tener en cuenta y la experimentación con problemas de tamaños mucho mayores también es un área que habría que explorar y explotar en el futuro.

Apéndice A

Descripción de las Funciones de Test

En esta tesis doctoral se han utilizado cinco problemas de test definidos sobre cadenas binarias, la función trampa de Deb[1] (TRAP) [89], las funciones jerárquicas si y solo si[2] (H-IFF) [357] y XOR[3] (H-XOR) [356], el problema deceptivo[4] masivamente multimodal de Goldbert et al.[5] (MMDP) [154] y SAT, los cuales se describen a continuación.

A.1. TRAP

La función completamente deceptiva de Deb de 4 bits (TRAP) tiene un único óptimo global rodeado por soluciones de baja calidad (fitness bajo) y un óptimo local rodeado por soluciones que se van alejando del óptimo local. Los métodos basados en gradiente habitualmente se ven confundidos y acaban siguiendo el camino hacia el óptimo local en vez de alcanzar el global. En términos matemáticos, la función TRAP se define como:

$$f(b_1 \cdots b_4) = \begin{cases} 0{,}6 - 0{,}2 \cdot u(b_1 \cdots b_4) & \text{si } u(b_1 \cdots b_4) < 4 \\ 1 & \text{si } u(b_1 \cdots b_4) = 4 \end{cases} \tag{A.1}$$

donde $u(s_1 \cdots s_i) = \sum_j s_j$ es el número de unos de la cadena binaria. Según esta definición, hay un único óptimo global (1111) en una región bastante aislada y un óptimo local (0000) rodeado por soluciones que van empeorando conforme más se alejan de este óptimo local. Esta función se usa como un bloque básico para cons-

[1] *4-bit fully deceptive function*

[2] *Hierarchical-if-and-only-if*

[3] *Hierarchical-XOR*

[4] El término inglés *deceptive* significa engañoso, sin embargo por coherencia con la literatura del área en castellano se va a usar el término deceptivo/a para referirse a él, a pesar de que esta palabra no exista en nuestro idioma según la RAE

[5] *massively multimodal deceptive problem*

truir un problema de orden superior concatenando k bloques (o subproblemas) y definiendo el fitness de una cadena de $4k$ bits como

$$\text{TRAP}(b_1 \cdots b_{4k}) = \sum_{i=1}^{k} f(b_{4(i-1)+1} \cdots b_{4i})$$

En todos los experimentos realizados se ha considerado que hay $k = 32$ subproblemas, es decir, cadenas de 128 bits (y por lo tanto el óptimo se encuentra en $opt = 32$).

A.2. H-IFF

La función H-IFF es un problema epistático recursivo cuya estructura jerárquica requiere que el algoritmo de búsqueda vaya de combinaciones de búsqueda de bits a combinaciones de búsqueda de esquemas de mayor orden (por lo tanto, hay que estimular la capacidad del algoritmo para identificar y combinar los bloques de construcción buenos). Esta función se define sobre cadenas binarias de 2^k bits a través de dos funciones auxiliares:

$$f : \{0, 1, \bullet\} \rightarrow \{0, 1\}. \tag{A.2}$$
$$t : \{0, 1, \bullet\} \rightarrow \{0, 1, \bullet\}. \tag{A.3}$$

donde '$\bullet$' se usa como un valor nulo (*null*). Más concretamente, estas dos funciones se definen de la siguiente manera:

$$f(a, b) = \begin{cases} 1 & a = b \neq \bullet \\ 0 & \text{en otro caso} \end{cases} \tag{A.4}$$

$$t(a, b) = \begin{cases} a & a = b \\ \bullet & \text{en otro caso} \end{cases} \tag{A.5}$$

Intuitivamente, f se utiliza para puntuar la contribución de los bloques de construcción y t se utiliza para capturar su interacción. Estas dos funciones se utilizan de la siguiente forma:

$$\text{H-IFF}_k(b_1 \cdots b_n) = \sum_{i=1}^{n/2} f(b_{2i-1}, b_{2i}) + 2 \cdot \text{H-IFF}_{k-1}(b'_1, \cdots, b'_{n/2}) \tag{A.6}$$

donde

$$b'_i = t(b_{2i-1}, b_{2i}) \tag{A.7}$$

y

$$\text{H-IFF}_0(\cdot) = 1 \tag{A.8}$$

Esta definición recursiva hace que la estructura de dependencias tome la forma de un árbol jerárquico en el que los bloques de construcción grandes se mueven hacia las posiciones superiores en el árbol y el peso de interacción va siendo cada vez mayor. Esta función tiene dos óptimos globales: $11\cdots1$ y $00\cdots0$. En todos los problemas se ha considerado que $k = 7$ (i.e., cadenas de 128 bits y por lo tanto el óptimo se encuentra en $opt = 576$).

A.3. H-XOR

La función Jerárquica-XOR (H-XOR) funciona de forma similar a H-IFF, pero se cambia la puntuación/interacción de los bloques de construcción de la siguiente manera:

$$f(a,b) = \begin{cases} 1 & a \neq b, \bullet \notin \{a,b\} \\ 0 & \text{en otro caso} \end{cases} \tag{A.9}$$

$$t(a,b) = \begin{cases} a & a \neq b, \bullet \notin \{a,b\} \\ \bullet & \text{en otro caso} \end{cases} \tag{A.10}$$

De esta forma se buscan las soluciones cuyas mitades son más diferentes entre sí, repitiendo el mismo proceso de forma recursiva sobre cada una de estas mitades, i.e., la solución óptima x_k^* para 2^k bits es $x_{k-1}^* \bar{x}_{k-1}^*$, donde $\bar{x}$ es el complemento bit a bit de x y $x_i^* \in \{0,1\}$. La consecuencia de esta estructura es que los memes útiles deben representarse sobre patrones más complejos y por lo tanto pasar por un escenario de optimización más duro. Se ha considerado que $k = 7$ (i.e., cadenas de 128 bits y por lo tanto el óptimo se encuentra en $opt = 576$).

A.4. MMDP

El problema deceptivo masivamente multimodal (MMDP) es una función deceptiva bipolar basada en bloques con dos óptimos globales localizados en posiciones extremas (y por lo tanto separados el uno del otro), con un atractor deceptivo local a medio camino entre ambos. Esta localización del atractor deceptivo provoca que sea mucho más fácil encontrar el óptimo local que el global (i.e., hay $\binom{L}{L/2}$ óptimos locales vs 2 óptimos globales en cada bloque, donde L es el número de bits de cada bloque). El MMDP básico se define para bloques de 6 bits de la siguiente manera:

$$f(b_1\cdots b_6) = \begin{cases} 1 & u(b_1\cdots b_6) \in \{0,6\} \\ 0 & u(b_1\cdots b_6) \in \{1,5\} \\ 0{,}360384 & u(b_1\cdots b_6) \in \{2,4\} \\ 0{,}640576 & u(b_1\cdots b_6) = 3 \end{cases} \tag{A.11}$$

En todos los problemas se han concatenado k copias de este bloque básico para crear un problema más complejo, concretamente se ha considerado $k = 24$ (i.e. cadenas de 144 bits y por consiguiente el óptimo global se encuentra en $opt = 24$).

A.5. SAT

El problema de la satisfacibilidad lógica fue el primer problema identificado como **NP**-completo [68] en el cual debe encontrarse una asignación verdadera para n variables que satisfagan una fórmula lógica determinada Φ. Más concretamente, se trata de un problema donde el objetivo es determinar si una expresión con n variables y sin cuantificadores tiene asociada una asignación de valores para sus variables que hace que la expresión sea verdadera. Por ejemplo dada una instancia de SAT con 5 variables, 4 cláusulas y 3 variables por cláusula tal como la siguiente:

$$(x_1 \vee \neg x_3 \vee x_5) \wedge (\neg x_2 \vee x_3 \vee \neg x_4) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_4 \vee x_5) \quad \text{(A.12)}$$

se trata de encontrar los valores de x_1, x_2, x_3, x_4, x_5 que la hacen verdadera. Se considera que esta fórmula se expresa en forma normal conjuntiva. Para los test realizados se han utilizado $n = 128$ variables y $k = 3$ variables por cláusula utilizando un generador de problemas para crear diferentes instancias satisfacibles con el ratio cláusulas/variables crítico ($opt = m = 4{,}3n = 550$) en cada ejecución del MMA.

Apéndice B

Datos Numéricos

B.1. Datos Numéricos Capítulo 4

A continuación se muestran los datos numéricos correspondientes a la experimentación descrita en la Sección 4.2 sobre las diferentes estrategias de reactivación de los nodos en función de las topologías de interconexión analizadas y sobre diversos escenarios de volatilidad. En las Tablas B.1-B.4 se muestran los resultados para el GA y en las Tablas B.5-B.8 para el MMA

Tabla B.1: Resultados (promediados para 20 ejecuciones) de los diferentes GAs basados en islas para los cuatro problemas considerados (no acción). Se muestra el número de veces que se encuentra el óptimo (n_{opt}), la mediana ($\tilde{x}$), la media ($\bar{x}$) y el error estándar de la media (σ_x). Tanto en esta tabla como en las siguientes, un círculo negro (•) indica que los resultados correspondientes son significativamente diferentes (usando el Test de Wilcoxon para $\alpha = 0{,}05$) con respecto a los resultados para $k = \infty$.

		TRAP				H-IFF			
topology	k	n_{opt}	$\tilde{x}$	$\bar{x} \pm \sigma_x$		n_{opt}	$\tilde{x}$	$\bar{x} \pm \sigma_x$	
Ring	∞	0	28.4	28.4 ± 0.1		1	397.0	407.9 ± 10.1	
	20	0	28.0	28.1 ± 0.2		0	371.0	377.2 ± 5.5	•
	10	0	26.2	26.3 ± 0.3	•	0	325.0	328.2 ± 5.0	•
	5	0	25.1	25.3 ± 0.3	•	0	298.0	302.9 ± 3.7	•
VN	∞	0	30.0	30.0 ± 0.1		0	436.0	433.0 ± 6.5	
	20	0	30.0	30.2 ± 0.2		0	414.0	420.2 ± 6.7	
	10	0	28.8	28.8 ± 0.2	•	0	384.0	384.5 ± 5.9	•
	5	0	26.7	26.5 ± 0.2	•	0	326.5	330.9 ± 7.2	•
HC	∞	0	30.3	30.3 ± 0.1		0	436.0	433.2 ± 6.6	
	20	0	30.3	30.1 ± 0.1		0	428.0	429.8 ± 7.4	
	10	0	28.8	28.6 ± 0.1	•	0	368.0	372.1 ± 7.9	•
	5	0	26.6	26.3 ± 0.3	•	0	323.0	326.8 ± 5.0	•
SF_1	∞	0	29.0	28.9 ± 0.1		0	397.0	403.6 ± 7.6	
	20	0	28.4	28.5 ± 0.2		0	380.0	383.6 ± 6.9	
	10	0	26.8	26.5 ± 0.4	•	0	326.0	328.1 ± 5.8	•
	5	0	25.0	25.0 ± 0.3	•	0	309.0	311.4 ± 5.8	•
SF_2	∞	0	29.8	29.8 ± 0.1		0	402.0	413.4 ± 8.1	
	20	0	29.9	29.7 ± 0.2		0	400.0	411.0 ± 7.7	
	10	0	28.0	27.9 ± 0.3	•	0	362.0	361.4 ± 5.6	•
	5	0	26.3	26.5 ± 0.3	•	0	340.0	333.8 ± 6.9	•

		H-XOR				MMDP			
topology	k	n_{opt}	$\tilde{x}$	$\bar{x} \pm \sigma_x$		n_{opt}	$\tilde{x}$	$\bar{x} \pm \sigma_x$	
Ring	∞	0	389.0	394.8 ± 5.8		0	20.0	20.2 ± 0.1	
	20	0	364.0	370.3 ± 5.7	•	0	20.0	19.9 ± 0.2	
	10	0	319.0	332.8 ± 6.9	•	0	18.5	18.5 ± 0.2	•
	5	0	310.0	309.7 ± 4.2	•	0	17.9	17.9 ± 0.2	•
VN	∞	0	436.0	435.9 ± 7.0		0	21.2	21.4 ± 0.1	
	20	1	422.0	426.4 ± 9.8		0	21.5	21.5 ± 0.1	
	10	0	371.0	380.9 ± 7.7	•	0	20.4	20.3 ± 0.1	•
	5	0	332.0	329.6 ± 5.1	•	0	19.0	18.8 ± 0.2	•
HC	∞	0	420.0	425.8 ± 7.0		0	21.5	21.6 ± 0.1	
	20	0	408.0	410.9 ± 6.7		0	21.5	21.3 ± 0.1	
	10	0	379.0	378.9 ± 5.9	•	0	20.2	20.2 ± 0.2	•
	5	0	320.0	326.4 ± 5.3	•	0	18.8	18.7 ± 0.2	•
SF_1	∞	0	397.0	407.7 ± 8.7		0	20.0	20.2 ± 0.1	
	20	0	368.0	377.7 ± 6.2	•	0	20.0	19.9 ± 0.2	
	10	0	342.0	341.3 ± 6.3	•	0	18.6	18.5 ± 0.3	•
	5	0	307.0	307.4 ± 4.2	•	0	18.2	17.9 ± 0.2	•
SF_2	∞	0	414.0	417.9 ± 5.2		0	21.1	21.3 ± 0.1	
	20	0	398.0	402.4 ± 4.5	•	0	21.1	21.1 ± 0.2	
	10	0	349.0	358.2 ± 6.0	•	0	20.0	20.1 ± 0.2	•
	5	0	327.0	328.6 ± 5.1	•	0	18.4	18.6 ± 0.2	•

Tabla B.2: Resultados (promediados para 20 ejecuciones) de los diferentes GAs basados en islas para los cuatro problemas considerados (checkpoint). Se muestra el número de veces que se encuentra el óptimo (n_{opt}), la mediana ($\tilde{x}$), la media ($\bar{x}$) y el error estándar de la media (σ_x).

		TRAP				H-IFF			
topology	k	n_{opt}	$\tilde{x}$	$\bar{x} \pm \sigma_x$		n_{opt}	$\tilde{x}$	$\bar{x} \pm \sigma_x$	
Ring	∞	0	28.4	28.4 ± 0.1		1	397.0	407.9 ± 10.1	
	20	0	28.4	28.4 ± 0.1		0	379.0	381.1 ± 5.3	•
	10	0	28.0	27.9 ± 0.2	•	0	365.5	368.0 ± 5.0	•
	5	0	27.4	27.5 ± 0.2	•	0	362.5	361.1 ± 4.4	•
VN	∞	0	30.0	30.0 ± 0.1		0	436.0	433.0 ± 6.5	
	20	0	30.0	30.2 ± 0.1		0	412.0	416.8 ± 7.0	
	10	0	30.0	29.7 ± 0.2		0	408.0	417.6 ± 6.3	
	5	0	29.6	29.6 ± 0.1		0	402.0	403.5 ± 5.7	•
HC	∞	0	30.3	30.3 ± 0.1		0	436.0	433.2 ± 6.6	
	20	0	30.4	30.1 ± 0.1		0	426.0	428.0 ± 6.1	
	10	0	30.0	30.0 ± 0.1		0	406.0	411.8 ± 5.6	•
	5	0	29.2	29.3 ± 0.1	•	1	391.0	403.9 ± 10.2	•
SF_1	∞	0	29.0	28.9 ± 0.1		0	397.0	403.6 ± 7.6	
	20	0	28.4	28.5 ± 0.2	•	0	381.5	387.7 ± 6.9	
	10	0	28.0	28.1 ± 0.2	•	0	368.0	373.9 ± 7.8	•
	5	0	27.6	27.6 ± 0.1	•	0	359.0	365.8 ± 5.6	•
SF_2	∞	0	29.8	29.8 ± 0.1		0	402.0	413.4 ± 8.1	
	20	0	29.6	29.7 ± 0.2		0	405.0	414.7 ± 8.0	
	10	0	29.6	29.3 ± 0.2		0	404.0	408.1 ± 6.2	
	5	0	29.0	29.1 ± 0.1	•	0	392.5	386.5 ± 5.1	•

		H-XOR				MMDP			
topology	k	n_{opt}	$\tilde{x}$	$\bar{x} \pm \sigma_x$		n_{opt}	$\tilde{x}$	$\bar{x} \pm \sigma_x$	
Ring	∞	0	389.0	394.8 ± 5.8		0	20.0	20.2 ± 0.1	
	20	0	368.0	371.3 ± 3.5	•	0	20.1	20.0 ± 0.2	
	10	0	369.0	371.6 ± 4.9	•	0	19.7	19.6 ± 0.1	•
	5	0	368.0	368.9 ± 5.0	•	0	19.3	19.5 ± 0.1	•
VN	∞	0	436.0	435.9 ± 7.0		0	21.2	21.4 ± 0.1	
	20	1	418.0	425.1 ± 10.0		0	21.5	21.5 ± 0.1	
	10	1	400.0	403.3 ± 10.8	•	0	21.1	21.1 ± 0.1	
	5	0	408.0	415.1 ± 5.8	•	0	20.8	20.9 ± 0.1	•
HC	∞	0	420.0	425.8 ± 7.0		0	21.5	21.6 ± 0.1	
	20	0	408.0	412.0 ± 5.0		0	21.5	21.4 ± 0.1	
	10	0	408.0	407.1 ± 5.1		0	21.1	21.1 ± 0.1	
	5	0	404.0	406.1 ± 4.2	•	0	20.8	20.9 ± 0.1	•
SF_1	∞	0	397.0	407.7 ± 8.7		0	20.0	20.2 ± 0.1	
	20	0	374.5	379.1 ± 6.6	•	0	19.9	19.9 ± 0.1	
	10	0	368.0	378.4 ± 6.5	•	0	20.0	19.8 ± 0.2	
	5	0	370.5	371.1 ± 6.2	•	0	19.3	19.6 ± 0.1	•
SF_2	∞	0	414.0	417.9 ± 5.2		0	21.1	21.3 ± 0.1	
	20	0	392.0	400.7 ± 5.0	•	0	21.5	21.1 ± 0.2	
	10	0	389.0	398.6 ± 6.7	•	0	20.8	20.8 ± 0.2	•
	5	0	395.0	390.3 ± 4.8	•	0	20.4	20.5 ± 0.1	•

Tabla B.3: Resultados (promediados para 20 ejecuciones) de los diferentes GAs basados en islas para los cuatro problemas considerados (reinicialización aleatoria). Se muestra el número de veces que se encuentra el óptimo (n_{opt}), la mediana ($\tilde{x}$), la media ($\bar{x}$) y el error estándar de la media (σ_x).

		TRAP				H-IFF			
topology	k	n_{opt}	$\tilde{x}$	$\bar{x} \pm \sigma_x$		n_{opt}	$\tilde{x}$	$\bar{x} \pm \sigma_x$	
Ring	∞	0	28.4	28.4 ± 0.1		1	397.0	407.9 ± 10.1	
	20	0	28.3	28.4 ± 0.1		0	380.0	381.1 ± 4.6	•
	10	0	27.3	27.2 ± 0.2	•	0	353.0	351.1 ± 6.0	•
	5	0	24.9	25.0 ± 0.4	•	0	315.0	318.4 ± 7.2	•
VN	∞	0	30.0	30.0 ± 0.1		0	436.0	433.0 ± 6.5	
	20	0	30.1	30.2 ± 0.1		0	406.0	414.7 ± 7.1	
	10	0	29.7	29.7 ± 0.2		0	412.0	414.7 ± 8.0	•
	5	0	28.2	28.4 ± 0.2	•	0	369.0	374.5 ± 7.9	•
HC	∞	0	30.3	30.3 ± 0.1		0	436.0	433.2 ± 6.6	
	20	0	30.2	30.1 ± 0.1		0	416.0	425.8 ± 6.7	
	10	0	29.6	29.5 ± 0.2	•	0	402.0	407.3 ± 6.6	•
	5	0	28.4	28.4 ± 0.3	•	0	370.0	366.9 ± 7.2	•
SF_1	∞	0	29.0	28.9 ± 0.1		0	397.0	403.6 ± 7.6	
	20	0	28.4	28.5 ± 0.2		0	380.0	383.1 ± 6.0	
	10	0	27.6	27.3 ± 0.3	•	0	343.5	347.1 ± 6.4	•
	5	0	26.2	25.7 ± 0.4	•	0	332.0	324.4 ± 7.0	•
SF_2	∞	0	29.8	29.8 ± 0.1		0	402.0	413.4 ± 8.1	
	20	0	29.8	29.7 ± 0.2		0	400.0	412.6 ± 8.6	
	10	0	28.6	28.9 ± 0.2	•	0	388.0	392.4 ± 6.4	•
	5	0	27.2	27.0 ± 0.4	•	0	350.0	347.3 ± 8.5	•

		H-XOR				MMDP			
topology	k	n_{opt}	$\tilde{x}$	$\bar{x} \pm \sigma_x$		n_{opt}	$\tilde{x}$	$\bar{x} \pm \sigma_x$	
Ring	∞	0	389.0	394.8 ± 5.8		0	20.0	20.2 ± 0.1	
	20	0	365.0	370.9 ± 5.3	•	0	20.0	19.9 ± 0.1	
	10	0	345.0	350.9 ± 6.5	•	0	19.3	19.3 ± 0.1	•
	5	0	319.0	317.9 ± 6.4	•	0	18.5	18.3 ± 0.2	•
VN	∞	0	436.0	435.9 ± 7.0		0	21.2	21.4 ± 0.1	
	20	0	422.0	422.4 ± 6.4		0	21.5	21.4 ± 0.1	
	10	0	390.0	392.6 ± 5.3	•	0	21.1	21.0 ± 0.1	
	5	0	370.0	367.2 ± 3.0	•	0	20.1	20.3 ± 0.1	•
HC	∞	0	420.0	425.8 ± 7.0		0	21.5	21.6 ± 0.1	
	20	0	408.0	417.2 ± 5.6		0	21.5	21.4 ± 0.1	
	10	0	396.0	395.8 ± 5.8	•	0	21.0	21.0 ± 0.2	•
	5	0	371.0	368.4 ± 6.4	•	0	20.4	20.1 ± 0.2	•
SF_1	∞	0	397.0	407.7 ± 8.7		0	20.0	20.2 ± 0.1	
	20	0	376.0	382.2 ± 7.2	•	0	19.9	19.9 ± 0.1	
	10	0	357.5	357.6 ± 6.2	•	0	19.3	19.4 ± 0.2	•
	5	0	337.5	337.9 ± 7.3	•	0	18.3	18.5 ± 0.2	•
SF_2	∞	0	414.0	417.9 ± 5.2		0	21.1	21.3 ± 0.1	
	20	0	392.0	400.8 ± 5.5	•	0	21.3	21.2 ± 0.2	
	10	0	382.0	386.4 ± 7.6	•	0	20.9	20.7 ± 0.3	
	5	0	357.0	349.6 ± 8.0	•	0	19.7	19.5 ± 0.2	•

Tabla B.4: Resultados (promediados para 20 ejecuciones) de los diferentes GAs basados en islas para los cuatro problemas considerados (reinicialización probabilística). Se muestra el número de veces que se encuentra el óptimo (n_{opt}), la mediana ($\tilde{x}$), la media ($\bar{x}$) y el error estándar de la media (σ_x).

		TRAP				H-IFF			
topology	k	n_{opt}	$\tilde{x}$	$\bar{x} \pm \sigma_x$		n_{opt}	$\tilde{x}$	$\bar{x} \pm \sigma_x$	
Ring	∞	0	28.4	28.4 ± 0.1		1	397.0	407.9 ± 10.1	
	20	0	28.3	28.4 ± 0.2		0	379.0	380.4 ± 4.8	•
	10	0	27.6	27.5 ± 0.2	•	0	358.0	361.6 ± 5.4	•
	5	0	26.0	26.1 ± 0.3	•	0	332.0	341.4 ± 5.1	•
VN	∞	0	30.0	30.0 ± 0.1		0	436.0	433.0 ± 6.5	
	20	0	30.1	30.2 ± 0.1		0	408.0	414.5 ± 6.8	
	10	0	30.0	29.8 ± 0.2		0	406.0	414.4 ± 8.5	•
	5	0	29.0	28.8 ± 0.1	•	0	395.0	390.4 ± 6.7	•
HC	∞	0	30.3	30.3 ± 0.1		0	436.0	433.2 ± 6.6	
	20	0	30.0	30.1 ± 0.1		0	424.0	431.0 ± 7.4	
	10	0	29.8	29.7 ± 0.2	•	0	401.0	415.7 ± 9.0	
	5	0	29.2	29.1 ± 0.2	•	0	377.0	380.6 ± 4.5	•
SF_1	∞	0	29.0	28.9 ± 0.1		0	397.0	403.6 ± 7.6	
	20	0	28.4	28.5 ± 0.2		0	382.0	382.4 ± 5.4	•
	10	0	27.6	27.5 ± 0.3	•	0	354.0	355.8 ± 8.2	•
	5	0	26.9	26.4 ± 0.3	•	0	343.0	343.3 ± 6.4	•
SF_2	∞	0	29.8	29.8 ± 0.1		0	402.0	413.4 ± 8.1	
	20	0	29.7	29.6 ± 0.2		0	404.0	415.9 ± 8.3	
	10	0	29.2	29.0 ± 0.2	•	0	386.0	392.8 ± 6.4	
	5	0	28.0	28.1 ± 0.1	•	0	370.0	368.7 ± 4.8	•

		H-XOR				MMDP			
topology	k	n_{opt}	$\tilde{x}$	$\bar{x} \pm \sigma_x$		n_{opt}	$\tilde{x}$	$\bar{x} \pm \sigma_x$	
Ring	∞	0	389.0	394.8 ± 5.8		0	20.0	20.2 ± 0.1	
	20	0	369.0	377.6 ± 5.6	•	0	20.0	20.0 ± 0.1	
	10	0	362.5	358.0 ± 6.4	•	0	19.3	19.4 ± 0.2	•
	5	0	337.0	336.9 ± 3.8	•	0	18.6	18.5 ± 0.2	•
VN	∞	0	436.0	435.9 ± 7.0		0	21.2	21.4 ± 0.1	
	20	1	428.0	430.4 ± 9.7		0	21.5	21.5 ± 0.1	
	10	0	402.0	400.1 ± 6.4	•	0	20.8	21.0 ± 0.1	•
	5	0	380.0	387.2 ± 4.3	•	0	20.8	20.7 ± 0.1	•
HC	∞	0	420.0	425.8 ± 7.0		0	21.5	21.6 ± 0.1	
	20	0	408.0	414.4 ± 5.5		0	21.3	21.3 ± 0.1	
	10	0	390.0	393.6 ± 6.2	•	0	21.1	21.0 ± 0.2	•
	5	0	384.0	380.1 ± 5.6	•	0	20.4	20.4 ± 0.1	•
SF_1	∞	0	397.0	407.7 ± 8.7		0	20.0	20.2 ± 0.1	
	20	0	371.5	381.7 ± 6.4	•	0	20.0	19.9 ± 0.2	
	10	0	360.0	364.4 ± 8.7	•	0	19.7	19.4 ± 0.2	•
	5	0	358.5	351.3 ± 6.0	•	0	18.4	18.7 ± 0.2	•
SF_2	∞	0	414.0	417.9 ± 5.2		0	21.1	21.3 ± 0.1	
	20	0	392.0	399.8 ± 4.6	•	0	21.5	21.1 ± 0.2	
	10	0	384.0	385.7 ± 5.9	•	0	20.8	20.7 ± 0.2	•
	5	0	362.0	361.5 ± 3.9	•	0	20.1	20.1 ± 0.1	•

Tabla B.5: Resultados (promediados para 20 ejecuciones) de los diferentes MMAs basados en islas para los cuatro problemas considerados (no acción). Se muestra el número de veces que se encuentra el óptimo (n_{opt}), la mediana ($\tilde{x}$), la media ($\bar{x}$) y el error estándar de la media (σ_x).

		TRAP				H-IFF			
topology	k	n_{opt}	$\tilde{x}$	$\bar{x} \pm \sigma_x$		n_{opt}	$\tilde{x}$	$\bar{x} \pm \sigma_x$	
Ring	∞	17	32.0	31.8 ± 0.1		19	576.0	570.4 ± 5.6	
	20	19	32.0	31.8 ± 0.2		19	576.0	570.0 ± 6.0	
	10	17	32.0	30.0 ± 0.6	•	17	456.0	451.6 ± 20.9	•
	5	12	28.1	28.6 ± 0.7	•	12	387.0	411.1 ± 22.0	•
VN	∞	17	32.0	31.8 ± 0.1		18	576.0	565.6 ± 7.2	
	20	19	32.0	31.9 ± 0.1		15	576.0	550.8 ± 10.4	
	10	17	32.0	31.5 ± 0.2		18	576.0	556.6 ± 11.9	
	5	13	30.8	29.9 ± 0.5	•	12	456.0	470.0 ± 18.2	•
HC	∞	16	32.0	31.6 ± 0.2		16	576.0	555.2 ± 9.6	
	20	19	32.0	31.9 ± 0.1		18	576.0	565.2 ± 7.5	
	10	17	32.0	31.7 ± 0.2		17	576.0	542.8 ± 15.6	
	5	14	32.0	30.6 ± 0.5		11	456.0	472.3 ± 20.2	•
SF_1	∞	19	32.0	31.9 ± 0.1		19	576.0	568.8 ± 7.2	
	20	17	32.0	31.7 ± 0.2		17	576.0	554.4 ± 12.5	
	10	17	32.0	30.3 ± 0.6	•	14	576.0	499.7 ± 23.0	•
	5	6	27.9	28.0 ± 0.7	•	11	374.5	396.9 ± 21.7	•
SF_2	∞	19	32.0	32.0 ± 0.0		19	576.0	570.0 ± 6.0	
	20	18	32.0	31.8 ± 0.2		17	576.0	553.0 ± 10.7	
	10	16	32.0	31.1 ± 0.4		14	496.0	506.6 ± 15.5	•
	5	13	32.0	29.6 ± 0.7	•	13	472.0	479.9 ± 22.4	•

		H-XOR				MMDP			
topology	k	n_{opt}	$\tilde{x}$	$\bar{x} \pm \sigma_x$		n_{opt}	$\tilde{x}$	$\bar{x} \pm \sigma_x$	
Ring	∞	0	402.5	408.1 ± 7.0		7	23.6	23.0 ± 0.3	
	20	0	395.0	397.2 ± 6.5		12	24.0	23.4 ± 0.2	
	10	0	351.0	354.9 ± 5.5	•	8	20.8	21.0 ± 0.5	•
	5	0	319.0	322.9 ± 5.4	•	2	19.0	20.2 ± 0.5	•
VN	∞	0	446.0	444.0 ± 7.5		12	24.0	23.5 ± 0.2	
	20	1	436.0	440.3 ± 9.9		15	24.0	23.6 ± 0.2	
	10	0	400.0	403.1 ± 7.0	•	11	23.6	23.1 ± 0.3	
	5	0	332.0	336.9 ± 6.1	•	2	21.1	21.3 ± 0.4	•
HC	∞	1	440.5	447.9 ± 8.6		13	24.0	23.5 ± 0.2	
	20	1	436.0	440.7 ± 8.6		14	24.0	23.6 ± 0.2	
	10	0	392.0	391.1 ± 7.1	•	7	22.9	22.4 ± 0.3	•
	5	0	339.5	338.8 ± 4.8	•	3	20.4	20.7 ± 0.4	•
SF_1	∞	1	396.0	412.9 ± 10.7		11	24.0	23.2 ± 0.3	
	20	0	405.0	406.6 ± 7.8		8	23.5	23.0 ± 0.3	
	10	0	355.0	356.6 ± 5.7	•	5	20.8	20.9 ± 0.4	•
	5	0	318.0	321.7 ± 8.0	•	0	18.6	19.2 ± 0.4	•
SF_2	∞	1	447.0	442.5 ± 8.7		14	24.0	23.6 ± 0.1	
	20	0	418.0	421.6 ± 6.7		13	24.0	23.6 ± 0.2	
	10	0	380.0	379.5 ± 6.7	•	8	23.5	22.5 ± 0.4	•
	5	0	344.5	346.9 ± 7.5	•	3	19.9	20.3 ± 0.5	•

Tabla B.6: Resultados (promediados para 20 ejecuciones) de los diferentes MMAs basados en islas para los cuatro problemas considerados (checkpoint). Se muestra el número de veces que se encuentra el óptimo (n_{opt}), la mediana ($\tilde{x}$), la media ($\bar{x}$) y el error estándar de la media (σ_x).

		TRAP				H-IFF		
topology	k	n_{opt}	$\tilde{x}$	$\bar{x} \pm \sigma_x$		n_{opt}	$\tilde{x}$	$\bar{x} \pm \sigma_x$
Ring	∞	17	32.0	31.8 ± 0.1		19	576.0	570.4 ± 5.6
	20	20	32.0	31.9 ± 0.1		18	576.0	564.8 ± 7.7
	10	18	32.0	31.6 ± 0.3		17	576.0	556.0 ± 11.4
	5	16	32.0	31.5 ± 0.2		18	576.0	559.0 ± 9.3
VN	∞	17	32.0	31.8 ± 0.1		18	576.0	565.6 ± 7.2
	20	19	32.0	31.9 ± 0.1		15	576.0	550.8 ± 10.4
	10	19	32.0	31.9 ± 0.1		18	576.0	565.2 ± 8.1
	5	18	32.0	31.9 ± 0.1		19	576.0	570.4 ± 5.6
HC	∞	16	32.0	31.6 ± 0.2		16	576.0	555.2 ± 9.6
	20	19	32.0	31.9 ± 0.1		17	576.0	559.6 ± 9.0
	10	18	32.0	31.9 ± 0.1		17	576.0	555.6 ± 11.4
	5	17	32.0	31.7 ± 0.1		18	576.0	565.2 ± 7.5
SF_1	∞	19	32.0	31.9 ± 0.1		19	576.0	568.8 ± 7.2
	20	17	32.0	31.7 ± 0.2		17	576.0	555.0 ± 12.0
	10	18	32.0	31.7 ± 0.2		18	576.0	558.4 ± 9.6
	5	17	32.0	31.7 ± 0.2		18	576.0	560.6 ± 8.8
SF_2	∞	19	32.0	32.0 ± 0.0		19	576.0	570.0 ± 6.0
	20	18	32.0	31.7 ± 0.2		17	576.0	558.6 ± 9.6
	10	17	32.0	31.7 ± 0.2		16	576.0	552.5 ± 12.0
	5	16	32.0	31.6 ± 0.2	•	18	576.0	563.2 ± 8.9

		H-XOR				MMDP		
topology	k	n_{opt}	$\tilde{x}$	$\bar{x} \pm \sigma_x$		n_{opt}	$\tilde{x}$	$\bar{x} \pm \sigma_x$
Ring	∞	0	402.5	408.1 ± 7.0		7	23.6	23.0 ± 0.3
	20	0	396.0	400.2 ± 5.9		10	23.8	23.5 ± 0.2
	10	0	382.0	385.4 ± 3.6	•	9	23.6	23.1 ± 0.3
	5	0	378.0	383.6 ± 5.3	•	11	23.8	22.8 ± 0.4
VN	∞	0	446.0	444.0 ± 7.5		12	24.0	23.5 ± 0.2
	20	1	447.0	444.3 ± 9.3		14	24.0	23.6 ± 0.2
	10	1	448.5	444.1 ± 10.2		16	24.0	23.7 ± 0.2
	5	0	428.0	428.8 ± 6.5		12	24.0	23.4 ± 0.2
HC	∞	1	440.5	447.9 ± 8.6		13	24.0	23.5 ± 0.2
	20	1	432.0	441.2 ± 8.2		16	24.0	23.7 ± 0.2
	10	1	418.0	431.4 ± 9.7	•	8	23.6	23.4 ± 0.2
	5	0	418.0	423.6 ± 6.5	•	10	23.6	23.3 ± 0.2
SF_1	∞	1	396.0	412.9 ± 10.7		11	24.0	23.2 ± 0.3
	20	0	400.0	407.8 ± 8.7		7	23.5	23.0 ± 0.3
	10	0	390.0	393.4 ± 5.2		8	23.1	22.6 ± 0.3
	5	0	376.0	379.8 ± 4.5	•	8	23.1	22.9 ± 0.3
SF_2	∞	1	447.0	442.5 ± 8.7		14	24.0	23.6 ± 0.1
	20	0	422.0	428.6 ± 6.5		14	24.0	23.7 ± 0.1
	10	0	401.0	405.3 ± 4.6	•	13	24.0	23.2 ± 0.3
	5	0	400.0	410.9 ± 8.0	•	11	23.8	23.4 ± 0.2

Tabla B.7: Resultados (promediados para 20 ejecuciones) de los diferentes MMAs basados en islas para los cuatro problemas considerados (reinicialización aleatoria). Se muestra el número de veces que se encuentra el óptimo (n_{opt}), la mediana ($\tilde{x}$), la media ($\bar{x}$) y el error estándar de la media (σ_x).

		TRAP				H-IFF			
topology	k	n_{opt}	$\tilde{x}$	$\bar{x} \pm \sigma_x$		n_{opt}	$\tilde{x}$	$\bar{x} \pm \sigma_x$	
Ring	∞	17	32.0	31.8 ± 0.1		19	576.0	570.4 ± 5.6	
	20	19	32.0	31.8 ± 0.1		19	576.0	570.0 ± 6.0	
	10	17	32.0	31.1 ± 0.4		16	576.0	536.6 ± 14.1	•
	5	13	28.1	28.1 ± 0.6	•	14	440.5	441.0 ± 22.9	•
VN	∞	17	32.0	31.8 ± 0.1		18	576.0	565.6 ± 7.2	
	20	19	32.0	31.9 ± 0.1		15	576.0	550.8 ± 10.4	
	10	19	32.0	31.9 ± 0.1		18	576.0	565.7 ± 7.6	
	5	17	32.0	31.6 ± 0.2		17	576.0	557.0 ± 10.5	
HC	∞	16	32.0	31.6 ± 0.2		16	576.0	555.2 ± 9.6	
	20	19	32.0	31.9 ± 0.1		16	576.0	554.8 ± 9.8	
	10	17	32.0	31.8 ± 0.1		17	576.0	555.3 ± 11.5	
	5	15	32.0	31.0 ± 0.5		18	576.0	555.6 ± 11.3	
SF_1	∞	19	32.0	31.9 ± 0.1		19	576.0	568.8 ± 7.2	
	20	17	32.0	31.7 ± 0.2		17	576.0	555.6 ± 11.5	
	10	19	32.0	31.2 ± 0.4	•	14	576.0	522.2 ± 15.9	•
	5	12	29.2	28.6 ± 0.7	•	14	576.0	500.6 ± 19.9	•
SF_2	∞	19	32.0	32.0 ± 0.0		19	576.0	570.0 ± 6.0	
	20	18	32.0	31.8 ± 0.2		17	576.0	558.6 ± 9.6	
	10	17	32.0	31.7 ± 0.2		15	576.0	545.5 ± 13.2	
	5	14	31.4	29.7 ± 0.6	•	17	576.0	503.9 ± 22.4	•

		H-XOR				MMDP			
topology	k	n_{opt}	$\tilde{x}$	$\bar{x} \pm \sigma_x$		n_{opt}	$\tilde{x}$	$\bar{x} \pm \sigma_x$	
Ring	∞	0	402.5	408.1 ± 7.0		7	23.6	23.0 ± 0.3	
	20	0	396.0	398.7 ± 6.2		11	24.0	23.4 ± 0.2	
	10	0	371.0	371.4 ± 5.8	•	6	22.0	22.1 ± 0.4	
	5	0	338.0	338.1 ± 6.4	•	6	20.0	20.5 ± 0.5	•
VN	∞	0	446.0	444.0 ± 7.5		12	24.0	23.5 ± 0.2	
	20	1	450.0	445.9 ± 9.0		14	24.0	23.6 ± 0.2	
	10	0	428.0	422.2 ± 7.0		12	23.8	23.4 ± 0.2	
	5	0	388.5	394.9 ± 8.1	•	6	22.6	22.6 ± 0.3	•
HC	∞	1	440.5	447.9 ± 8.6		13	24.0	23.5 ± 0.2	
	20	1	436.0	445.3 ± 8.0		15	24.0	23.7 ± 0.2	
	10	0	408.0	416.1 ± 6.9	•	12	24.0	23.3 ± 0.3	
	5	0	380.0	378.8 ± 6.1	•	8	23.1	22.5 ± 0.4	•
SF_1	∞	1	396.0	412.9 ± 10.7		11	24.0	23.2 ± 0.3	
	20	0	392.0	404.1 ± 7.9		8	23.5	22.9 ± 0.3	
	10	0	378.0	387.3 ± 7.5	•	7	22.7	22.3 ± 0.4	
	5	0	337.5	344.4 ± 7.8	•	3	20.3	20.8 ± 0.4	•
SF_2	∞	1	447.0	442.5 ± 8.7		14	24.0	23.6 ± 0.1	
	20	0	412.0	419.6 ± 6.5	•	14	24.0	23.6 ± 0.2	
	10	0	393.0	394.7 ± 4.7	•	11	23.6	23.1 ± 0.3	
	5	0	360.0	357.4 ± 8.2	•	6	22.7	22.4 ± 0.4	•

Tabla B.8: Resultados (promediados para 20 ejecuciones) de los diferentes MMAs basados en islas para los cuatro problemas considerados (reinicialización probabilística). Se muestra el número de veces que se encuentra el óptimo (n_{opt}), la mediana ($\tilde{x}$), la media ($\bar{x}$) y el error estándar de la media (σ_x).

		TRAP				H-IFF			
topology	k	n_{opt}	$\tilde{x}$	$\bar{x} \pm \sigma_x$		n_{opt}	$\tilde{x}$	$\bar{x} \pm \sigma_x$	
Ring	∞	17	32.0	31.8 ± 0.1		19	576.0	570.4 ± 5.6	
	20	19	32.0	31.8 ± 0.1		19	576.0	570.0 ± 6.0	
	10	16	32.0	31.1 ± 0.5		16	576.0	528.3 ± 15.6	•
	5	16	32.0	30.6 ± 0.5	•	17	460.0	470.0 ± 22.1	•
VN	∞	17	32.0	31.8 ± 0.1		18	576.0	565.6 ± 7.2	
	20	19	32.0	31.9 ± 0.1		15	576.0	550.8 ± 10.4	
	10	19	32.0	31.9 ± 0.1		17	576.0	560.3 ± 9.2	
	5	19	32.0	32.0 ± 0.0		18	576.0	564.0 ± 8.3	
HC	∞	16	32.0	31.6 ± 0.2		16	576.0	555.2 ± 9.6	
	20	19	32.0	31.9 ± 0.1		16	576.0	554.8 ± 9.8	
	10	18	32.0	31.9 ± 0.1		16	576.0	552.8 ± 10.7	
	5	18	32.0	31.7 ± 0.2		13	576.0	537.6 ± 12.1	
SF_1	∞	19	32.0	31.9 ± 0.1		19	576.0	568.8 ± 7.2	
	20	17	32.0	31.7 ± 0.2		17	576.0	558.8 ± 9.4	
	10	17	32.0	30.9 ± 0.5		13	576.0	521.0 ± 17.3	•
	5	11	31.6	30.1 ± 0.5	•	14	520.0	498.9 ± 19.4	•
SF_2	∞	19	32.0	32.0 ± 0.0		19	576.0	570.0 ± 6.0	
	20	18	32.0	31.8 ± 0.2		17	576.0	558.6 ± 9.6	
	10	17	32.0	31.6 ± 0.2		14	576.0	544.2 ± 12.2	•
	5	17	32.0	31.4 ± 0.3		15	576.0	529.4 ± 16.1	•

		H-XOR				MMDP			
topology	k	n_{opt}	$\tilde{x}$	$\bar{x} \pm \sigma_x$		n_{opt}	$\tilde{x}$	$\bar{x} \pm \sigma_x$	
Ring	∞	0	402.5	408.1 ± 7.0		7	23.6	23.0 ± 0.3	
	20	0	396.0	400.6 ± 7.1		12	24.0	23.5 ± 0.2	
	10	0	377.0	377.2 ± 4.7	•	8	22.9	22.7 ± 0.3	
	5	0	355.0	354.4 ± 5.0	•	3	22.0	21.7 ± 0.4	•
VN	∞	0	446.0	444.0 ± 7.5		12	24.0	23.5 ± 0.2	
	20	1	449.0	444.4 ± 9.3		14	24.0	23.6 ± 0.2	
	10	0	422.0	429.1 ± 8.1		15	24.0	23.7 ± 0.1	
	5	0	400.0	416.2 ± 9.3	•	14	24.0	23.7 ± 0.1	
HC	∞	1	440.5	447.9 ± 8.6		13	24.0	23.5 ± 0.2	
	20	1	442.0	445.7 ± 8.2		15	24.0	23.7 ± 0.1	
	10	1	413.0	424.8 ± 10.3	•	11	24.0	23.5 ± 0.2	
	5	0	416.0	416.1 ± 6.8	•	6	22.7	22.5 ± 0.3	•
SF_1	∞	1	396.0	412.9 ± 10.7		11	24.0	23.2 ± 0.3	
	20	0	402.5	404.9 ± 7.6		9	23.5	23.1 ± 0.3	
	10	0	384.0	388.9 ± 6.9	•	8	22.7	22.3 ± 0.4	
	5	0	364.0	365.2 ± 5.8	•	5	20.4	20.9 ± 0.4	•
SF_2	∞	1	447.0	442.5 ± 8.7		14	24.0	23.6 ± 0.1	
	20	0	420.0	425.1 ± 6.3		14	24.0	23.7 ± 0.1	
	10	0	388.0	393.9 ± 5.7	•	10	23.8	23.2 ± 0.3	
	5	0	380.0	391.6 ± 6.3	•	9	23.1	22.9 ± 0.3	•

B.2. Datos Numéricos Capítulo 5

A continuación se muestran los datos numéricos correspondientes a la experimentación descrita en la Sección 5.2 sobre las diferentes topologías de interconexión entre los nodos analizadas y las estrategias de equilibrado estudiadas para los cuatro problemas de test considerados. En las Tablas B.9 y B.10 se muestran los resultados para los problemas TRAP y H-IFF y en las Tablas B.11 y B.12 para H-XOR y MMDP.

Tabla B.9: Desviación (en %) con respecto a la solución óptima (25 ejecuciones) de diferentes MMAs para TRAP y H-IFF con redes SF. Se muestra la mediana ($\tilde{x}$), la media ($\bar{x}$) y el error estándar de la media ($\sigma_{\bar{x}}$).

			TRAP		H-IFF	
topología	estrategia	k	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$
SF_1	noB	20	6.25	7.03 ± 0.93	0.00	14.97 ± 3.44
		10	18.13	19.00 ± 1.33	39.41	36.40 ± 3.06
		5	33.13	33.36 ± 1.20	56.94	55.19 ± 1.03
		2	48.75	49.15 ± 0.70	61.46	61.01 ± 0.45
		1	53.47	52.93 ± 0.64	65.10	65.06 ± 0.19
	LBQ	20	1.88	2.63 ± 0.60	0.00	7.88 ± 2.45
		10	6.87	9.15 ± 1.41	0.00	13.83 ± 3.55
		5	15.00	15.78 ± 1.88	21.53	22.70 ± 3.78
		2	40.00	39.58 ± 0.90	52.43	51.64 ± 1.61
		1	51.82	50.90 ± 0.64	63.72	63.25 ± 0.42
SF_2	noB	20	0.00	1.30 ± 0.36	0.00	5.17 ± 2.22
		10	9.38	9.35 ± 1.09	0.00	11.84 ± 2.90
		5	26.25	27.02 ± 1.51	47.40	44.85 ± 2.55
		2	47.50	47.54 ± 0.75	62.67	61.56 ± 0.68
		1	53.75	54.08 ± 0.59	64.58	64.52 ± 0.23
	LBQ	20	0.00	0.40 ± 0.19	0.00	4.44 ± 1.86
		10	0.00	0.92 ± 0.29	0.00	8.53 ± 2.61
		5	2.50	3.48 ± 0.65	0.00	10.88 ± 2.46
		2	21.88	22.83 ± 1.25	21.88	21.71 ± 3.39
		1	44.38	44.15 ± 1.04	51.39	50.17 ± 1.79
SF_4	noB	20	0.00	0.40 ± 0.19	0.00	5.94 ± 2.27
		10	1.25	2.73 ± 0.67	0.00	12.30 ± 3.05
		5	16.25	15.98 ± 1.40	24.48	22.50 ± 4.09
		2	46.88	46.35 ± 1.13	61.28	60.32 ± 0.64
		1	52.50	52.55 ± 0.69	64.24	63.85 ± 0.33
	LBQ	20	0.00	0.10 ± 0.07	0.00	2.61 ± 1.49
		10	0.00	0.55 ± 0.36	0.00	4.72 ± 1.77
		5	2.50	3.05 ± 0.49	0.00	10.11 ± 2.90
		2	15.79	15.86 ± 1.38	19.44	15.71 ± 3.43
		1	30.00	29.10 ± 1.42	24.48	23.80 ± 3.20

Tabla B.10: Desviación (en %) con respecto a la solución óptima (25 ejecuciones) de diferentes MMAs para TRAP y H-IFF con redes SW. Se muestra la mediana ($\tilde{x}$), la media ($\bar{x}$) y el error estándar de la media ($\sigma_{\bar{x}}$).

			TRAP		H-IFF	
topología	estrategia	k	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$
SW_1	noB	20	6.87	5.85 ± 0.66	0.00	11.06 ± 3.58
		10	14.38	15.85 ± 1.67	27.43	25.01 ± 4.10
		5	32.50	32.82 ± 1.38	55.56	53.65 ± 1.25
		2	48.75	47.59 ± 0.94	61.92	61.13 ± 0.76
		1	53.75	53.59 ± 0.54	64.93	64.69 ± 0.21
	LBQ	20	0.00	1.50 ± 0.50	0.00	4.43 ± 1.65
		10	6.87	7.92 ± 1.25	0.00	12.25 ± 3.17
		5	18.75	18.48 ± 2.00	31.60	32.44 ± 2.79
		2	41.87	40.55 ± 1.36	55.38	54.41 ± 1.27
		1	53.13	52.55 ± 0.56	64.24	63.81 ± 0.36
SW_2	noB	20	2.50	3.79 ± 0.95	0.00	6.56 ± 2.29
		10	8.13	7.75 ± 1.25	19.44	17.09 ± 3.27
		5	28.75	29.52 ± 1.47	42.19	34.11 ± 4.09
		2	47.50	46.25 ± 1.30	62.36	61.80 ± 0.44
		1	53.75	53.20 ± 0.54	64.93	64.76 ± 0.23
	LBQ	20	0.00	1.52 ± 0.49	0.00	6.97 ± 2.26
		10	1.25	2.05 ± 0.51	0.00	5.11 ± 1.89
		5	7.50	7.55 ± 0.89	0.00	10.85 ± 2.86
		2	35.62	35.30 ± 1.03	47.40	40.28 ± 2.91
		1	51.25	50.79 ± 0.70	63.54	63.31 ± 0.30
SW_4	noB	20	0.00	1.60 ± 0.53	16.67	12.08 ± 2.50
		10	3.75	4.20 ± 0.97	19.44	15.22 ± 3.19
		5	16.25	16.27 ± 1.58	22.05	23.84 ± 3.78
		2	45.63	44.93 ± 1.07	60.48	59.63 ± 1.19
		1	52.86	52.18 ± 0.56	64.76	64.46 ± 0.22
	LBQ	20	0.00	0.55 ± 0.28	0.00	5.67 ± 2.14
		10	0.00	1.35 ± 0.44	0.00	7.19 ± 2.30
		5	3.13	4.35 ± 0.88	0.00	6.39 ± 2.43
		2	21.25	22.93 ± 1.15	22.57	22.94 ± 3.70
		1	49.38	48.92 ± 0.65	54.34	53.42 ± 1.31

Tabla B.11: Desviación (en %) con respecto a la solución óptima (25 ejecuciones) de diferentes MMAs para H-XOR y MMDP con redes SF. Se muestra la mediana ($\tilde{x}$), la media ($\bar{x}$) y el error estándar de la media ($\sigma_{\bar{x}}$).

			H-XOR		MMDP	
topología	estrategia	k	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$
SF_1	noB	20	41.32	40.87 ± 0.77	8.99	10.30 ± 1.01
		10	48.44	49.30 ± 0.88	19.98	21.22 ± 1.17
		5	57.47	57.11 ± 0.46	28.45	29.29 ± 0.76
		2	63.07	62.77 ± 0.25	38.12	38.61 ± 0.43
		1	64.76	64.90 ± 0.15	40.79	40.41 ± 0.52
	LBQ	20	34.38	33.93 ± 1.04	4.49	4.99 ± 0.56
		10	38.02	39.15 ± 0.97	11.65	11.96 ± 1.04
		5	44.97	46.07 ± 1.26	19.14	19.75 ± 1.38
		2	58.33	57.89 ± 0.56	33.30	33.02 ± 0.85
		1	64.41	64.21 ± 0.23	40.46	40.32 ± 0.48
SF_2	noB	20	30.90	31.56 ± 1.03	4.49	4.90 ± 0.49
		10	40.10	40.47 ± 1.04	14.32	13.96 ± 1.10
		5	53.30	53.11 ± 0.91	27.79	27.46 ± 0.88
		2	62.50	62.45 ± 0.33	37.29	37.46 ± 0.62
		1	64.76	64.43 ± 0.25	41.63	41.14 ± 0.36
	LBQ	20	29.17	28.44 ± 1.20	3.00	2.56 ± 0.32
		10	31.94	32.26 ± 0.82	4.49	4.36 ± 0.52
		5	35.42	34.97 ± 0.97	8.66	7.98 ± 0.55
		2	47.22	47.03 ± 0.53	21.80	21.48 ± 0.87
		1	60.42	60.48 ± 0.54	36.78	36.38 ± 0.57
SF_4	noB	20	33.33	31.78 ± 0.96	1.50	1.36 ± 0.28
		10	35.42	34.40 ± 0.74	5.99	6.49 ± 0.64
		5	44.44	44.15 ± 0.86	16.47	17.03 ± 0.80
		2	61.28	61.22 ± 0.43	36.63	37.17 ± 0.65
		1	64.93	64.76 ± 0.18	40.80	40.71 ± 0.49
	LBQ	20	27.78	26.58 ± 1.10	0.00	1.24 ± 0.33
		10	31.25	30.08 ± 0.94	1.50	2.76 ± 0.50
		5	34.38	33.13 ± 0.84	5.99	6.52 ± 0.56
		2	43.40	43.79 ± 0.94	14.65	14.78 ± 0.78
		1	51.56	52.06 ± 0.67	26.48	25.79 ± 1.07

Tabla B.12: Desviación (en %) con respecto a la solución óptima (25 ejecuciones) de diferentes MMAs para H-XOR y MMDP con redes SW. Se muestra la mediana ($\tilde{x}$), la media ($\bar{x}$) y el error estándar de la media ($\sigma_{\bar{x}}$).

			H-XOR		MMDP	
topología	estrategia	k	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$
SW_1	noB	20	38.37	37.46 ± 1.05	10.15	10.16 ± 1.11
		10	47.92	48.01 ± 0.96	17.97	18.72 ± 1.44
		5	57.29	57.36 ± 0.56	29.62	29.57 ± 0.85
		2	63.02	62.96 ± 0.25	36.96	37.07 ± 0.62
		1	65.10	64.59 ± 0.28	40.79	41.45 ± 0.48
	LBQ	20	32.64	31.00 ± 1.77	5.99	7.85 ± 1.12
		10	35.59	36.40 ± 1.33	13.48	13.43 ± 1.32
		5	43.92	45.87 ± 1.18	19.98	21.02 ± 1.12
		2	59.38	59.10 ± 0.68	33.96	34.65 ± 0.76
		1	64.41	63.99 ± 0.29	41.24	41.31 ± 0.37
SW_2	noB	20	34.03	34.17 ± 1.26	4.49	4.65 ± 0.69
		10	39.76	40.22 ± 0.82	11.65	11.98 ± 1.11
		5	53.30	53.16 ± 1.06	26.30	26.28 ± 0.80
		2	63.02	62.48 ± 0.39	36.63	36.42 ± 0.71
		1	64.90	64.79 ± 0.17	41.12	41.08 ± 0.43
	LBQ	20	33.33	32.26 ± 0.84	4.49	4.75 ± 0.55
		10	35.76	35.44 ± 0.75	5.99	7.30 ± 0.74
		5	40.10	39.13 ± 0.79	11.65	11.34 ± 0.72
		2	54.69	54.48 ± 0.66	30.64	30.78 ± 0.61
		1	63.54	63.44 ± 0.30	40.79	40.66 ± 0.35
SW_4	noB	20	33.33	33.19 ± 1.05	1.50	2.38 ± 0.50
		10	38.89	38.84 ± 0.63	7.16	7.20 ± 0.81
		5	44.97	46.28 ± 0.90	17.64	17.72 ± 1.21
		2	61.63	61.28 ± 0.51	36.12	36.00 ± 0.75
		1	64.93	64.65 ± 0.21	40.79	40.21 ± 0.48
	LBQ	20	31.25	30.54 ± 0.98	1.50	2.43 ± 0.47
		10	34.72	33.51 ± 1.09	4.49	4.45 ± 0.52
		5	37.50	36.28 ± 0.82	8.66	8.57 ± 0.55
		2	48.61	48.68 ± 0.80	23.63	23.88 ± 0.87
		1	61.81	61.36 ± 0.55	38.46	38.15 ± 0.43

Bibliografía

[1] E.H.L. Aarts y J. Korst. *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing*. John Wiley & Sons, Chichester, England, 1989.

[2] S. G. Akl. *Parallel Computation Models and Methods*. Prentice Hall, 1997.

[3] E. Alba. Parallel evolutionary algorithms can achieve super-linear performance. *Information Processing Letters*, 82(1):7–13, 2002.

[4] E. Alba. *Parallel Metaheuristics: A New Class of Algorithms*. Wiley-Interscience, Hoboken, New Jersey, USA, 2005.

[5] E. Alba y C. Cotta. Evolutionary algorithms. En [283], págs. 3–19.

[6] E. Alba y B. Dorronsoro. Solving the vehicle routing problem by using cellular genetic algorithms. En J. Gottlieb y G.R. Raidl, eds., *Evolutionary Computation in Combinatorial Optimization*, tomo 3004 de *Lecture Notes in Computer Science*, págs. 11–20. Springer Verlag, Berlin Heidelberg, 2004.

[7] E. Alba y B. Dorronsoro. The exploration/exploitation tradeoff in dynamic cellular genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 9(2):126–142, 2005.

[8] 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, 2006.

[9] E. Alba y B. Dorronsoro. A hybrid cellular genetic algorithm for the capacitated vehicle routing problem. En A. Abraham, C. Grosan, y W. Pedrycz, eds., *Engineering Evolutionary Intelligent Systems*, tomo 82 de *Studies in Computational Intelligence*, págs. 379–422. Springer Verlag, Berlin Heidelberg, 2008.

[10] E. Alba, M. Giacobini, M. Tomassini, y S. Romero. Comparing synchronous and asynchronous cellular genetic algorithms. En [243], págs. 601–610.

[11] E. Alba y G. Luque. Growth curves and takeover time in evolutionary algorithms. En K. Deb, ed., *Genetic and Evolutionary Computation - GECCO*

2004, tomo 3102 de *Lecture Notes in Computer Science*, págs. 864–876. Springer Verlag, Berlin Heidelberg, 2004.

[12] E. Alba y M. Tomassini. Parallelism and evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 6(5):443–462, 2002.

[13] E. Alba y J.M. Troya. Influence of the migration policy in parallel distributed gas with structured and panmictic populations. *Applied Intelligence*, 12(3):163–181, 2000.

[14] M. Allen. Do-it-yourself climate prediction. *Nature*, 401:642, 1999.

[15] F. Almeida, D. González, y L.M. Moreno. The master-slave paradigm on heterogeneous systems: A dynamic programming approach for the optimal mapping. *Journal of Systems Architecture*, 52(2):105–116, 2006.

[16] B. Alpern y F.B. Schneider. Recognizing safety and liveness. *Distributed Computing*, 2(3):117–126, 1987.

[17] J.E. Amaya Salazar. *Modelos meméticos cooperativos para la optimización de problemas combinatorios*. Tesis Doctoral, Universidad de Málaga, 2011.

[18] R. Ananthanarayanan, M. Mohania, y A. Gupta. Management of conflicting obligations in self-protecting policy-based systems. En *Second International Conference on Autonomic Computing - ICAC 2005*, págs. 274–285. IEEE Computer Society, Seattle, WA, USA, 2005.

[19] E. Anceaume, X. Defago, M. Gradinariu, y M. Roy. Towards a theory of self-organization. En *9th International Conference On Principles Of Distributed Systems - OPODIS 2005*, tomo 3974 de *Lecture Notes in Computer Science*, págs. 191–205. Springer-Verlag, Berlin Heidelberg, 2005.

[20] D.P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, y D. Werthimer. SETI@home: an experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61, 2002.

[21] P.J. Angeline. Morphogenic evolutionary computations: Introduction, issues and example. En J.R. McDonnell et al., ed., *Fourth Annual Conference on Evolutionary Programming*, págs. 387–402. MIT Press, Cambridge, Massachusetts, 1995.

[22] L. Araujo y J.J. Merelo Guervós. Multikulti algorithm: Using genotypic differences in adaptive distributed evolutionary algorithm migration policies. En *IEEE Congress on Evolutionary Computation - CEC 2009*, págs. 2858–2865. IE, Trondheim, Norway, 2009.

[23] A. Arora y M. Gouda. Closure and convergence: a foundation of fault-tolerant computing. *IEEE Transactions on Software Engineering*, 19:1015–1027, 1993.

[24] S. Arora y B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 1 ed$^{\underline{ón}}$., 2009.

[25] M.D. Assunção, R.N. Calheiros, S. Bianchi, M.A.S. Netto, y R. Buyya. Big data computing and clouds: Trends and future directions. *Journal of Parallel and Distributed Computing*, 79-80(Special Issue on Scalable Systems for Big Data Management and Analytics):3–15, 2015.

[26] G. Ausiello, A. D' Atri, y M. Protasi. Structure preserving reductions among convex optimization problems. *Journal of Computer and System Sciences*, 21(1):136 – 153, 1980.

[27] O. Babaoglu, M. Jelasity, A. Montresor, C. Fetzer, S. Leonardi, A. van Moorsel, y M. van Steen, eds. *Self-star Properties in Complex Information Systems*, tomo 3460 de *Lecture Notes in Computer Science*. Springer-Verlag, Berlin Heidelberg, 2005.

[28] T. Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York, 1996.

[29] T. Bäck et al., eds. *Handbook of Evolutionary Computation*. Institute of Physics Publishing and Oxford University Press, Bristol, New York, 1997.

[30] S. Baluja. Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Inf. Téc. CMU-CS-95-19, Carnegie Mellon University, Pittsburgh, PA, USA, 1994.

[31] S. Baluja y S. Davies. Using optimal dependency-trees for combinatorial optimization: Learning the structure of the search space. En *14th International Conference on Machine Learning*, págs. 30–38. Morgan Kaufmann Publishers, 1997.

[32] C. Banino, O. Beaumont, L. Carter, J. Ferrante, A. Legrand, y Y. Robert. Scheduling strategies for master-slave tasking on heterogeneous processor platforms. *IEEE Transactions on Parallel and Distributed Systems*, 2004.

[33] W. Banzhaf et al., eds. *Genetic and Evolutionary Computation - GECCO 1999*, tomo 1. Morgan Kaufmann Publishers, Orlando, FL, 1999.

[34] A.L. Barabási y A. Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[35] A.L. Barabási, V.W. Freeh, H. Jeong, y J.B. Brockman. Parasitic computing. *Letters to Nature*, 412:894–897, 2001.

[36] D. Barmpoutis y R.M. Murray. Networks with the smallest average distance and the largest average clustering. *arXiv:1007.4031 [q-bio.MN]*, 2010.

[37] T. Bartz-Beielstein et al., eds. *Parallel Problem Solving from Nature - PPSN XIII*, tomo 8672 de *Lecture Notes in Computer Science*. Springer Verlag, Berlin Heidelberg, 2014.

[38] R. Battiti y G. Tecchiolli. The reactive tabu search. *ORSA Journal on Computing*, 6:126–140, 1994.

[39] O. Beaumont, A. Legrand, y Y. Robert. The master-slave paradigm with heterogeneous processors. *IEEE Transactions on Parallel and Distributed Systems*, 14(9):897–908, 2003.

[40] O. Beaumont, A. Legrand, y Y. Robert. Scheduling strategies for mixed data and task parallelism on heterogeneous clusters and grids. En *11th Euromicro Conference on Parallel, Distributed and Network-Based Processing - Euro-PDP 2003*, págs. 209–216. 2003.

[41] A. Berns y S. Ghosh. Dissecting self-$\star$ properties. En *Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems - SASO 2009*, págs. 10–19. IEEE Press, San Francisco, CA, 2009.

[42] H.G. Beyer. Toward a theory of evolution strategies: Self-adaptation. *Evolutionary Computation*, 3(3):311–348, 1995.

[43] S. Blackmore. *The Meme Machine*. Oxford University Press, 2000.

[44] G.S. Blair, G. Coulson, L. Blair, H. Duran-Limon, P. Grace, R. Moreira, y N. Parlavantzas. Reflection, self-awareness and self-healing in openORB. En *First Workshop on Self-healing Systems*, págs. 9–14. ACM Press, New York, NY, USA, 2002.

[45] C. Blum y A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.

[46] J.S.D. Bonet, C.L. Isbell Jr., y P. Viola. Mimic: Finding optima by estimating probability densities. En M. Mozer, M. Jordan, y T. Petsche, eds., *Advances in Neural Information Processing Systems*, tomo 9, págs. 424–430. MIT Press, 1996.

[47] A.G. Bronevich y W. Meyer. Load balancing algorithms based on gradient methods and their analysis through algebraic graph theory. *Journal of Parallel and Distributed Computing*, 68(2):209–220, 2008.

[48] E. Burke y A. Smith. Hybrid evolutionary techniques for the maintenance scheduling problem. *IEEE Transactions on Power Systems*, 15(1):122–128, 2000.

[49] E.K. Burke, G. Kendall, y E. Soubeiga. A tabu search hyperheuristic for timetabling and rostering. *Journal of Heristics*, 9(6):451–470, 2003.

[50] L. Cai, J. Chen, R.G. Downey, y M.R. Fellows. Advise classes of parameterized tractability. *Annals of Pure and Applied Logic*, 84:119–138, 1997.

[51] L. Cai, J. Chen, R.G. Downey, y M.R. Fellows. On the parameterized complexity of short computation and factorization. *Archive for Mathematical Logic*, 36:321–337, 1997.

[52] E. Cantú-Paz. *Efficient and Accurate Parallel Genetic Algorithms*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[53] E. Cantú-Paz. Migration policies, selection pressure, and parallel evolutionary algorithms. *Journal of Heuristics*, 7(4):311–334, 2001.

[54] E. Cantú-Paz y D. Goldberg. On the scalability of parallel genetic algorithms. *Journal Evolutionary Computation*, 7(4):429–449, 1999.

[55] A. Caponio, G.L. Cascella, F. Neri, N.Salvatore, y M. Sumner. A fast adaptive memetic algorithm for on-line and off-line control design of pmsm drives. *IEEE Transactions on Systems Man and Cybernetics Part B, Special Issue on Memetic Algorithms*, 37(1):28–41, 2007.

[56] M. Cesati. Compendium of parameterized problems. 2001. URL http://cesati.sprg.uniroma2.it/research/compendium/. Accessed 04/05/2015.

[57] B. Cestnik. Estimating probabilities: A crucial task in machine learning. En L. Aiello, ed., *9th European Conference on Artificial Intelligence*, págs. 147–149. Pitman, Stockholm, Sweden, 1990.

[58] K Chakhlevitch y P. Cowling. Hyperheuristics: Recent developments. En [76], págs. 3–29.

[59] J. Chen, I. Kanj, y W. Jia. Vertex cover: Further observations and further improvements. En P. Widmayer, G. Neyer, y S. Eidenbenz, eds., *25th International Workshop on Graph-Theoretic Concepts in Computer Science*, tomo 1665 de *Lecture Notes in Computer Science*, págs. 313–324. Springer-Verlag, Berlin Heidelberg, 1999.

[60] X. Chen y Y. S. Ong. A conceptual modeling of meme complexes in stochastic search. *IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews*, 42(5):612–625, 2012.

[61] X. Chen, Y.S. Ong, M.H. Lim, y K.C. Tan. A multi-facet survey on memetic computation. *IEEE Transactions on Evolutionary Computation*, 15(5):591–607, 2011.

[62] Y. Chen, J. Flum, y M. Grohe. An analysis of the W*-hierarchy. *The Journal of Symbolic Logic*, 72(2):513–534, 2007.

[63] J. Cheng, G. Zhang, y F. Neri. Enhancing distributed differential evolution with multicultural migration for global numerical optimization. *Information Sciences*, 247:72–93, 2013.

[64] D. Chiang y J. Moh. A global optimization method based on simulated annealing and evolutionary strategy. En D.S. Huang, K. Li, y G. Irwin, eds., *Intelligent Computing*, tomo 4113 de *Lecture Notes in Computer Science*, págs. 790–801. Springer-Verlag, Berlin Heidelberg, 2006.

[65] P.C. Chu y J.E. Beasley. A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics*, 4:63–86, 1998.

[66] N. Cole, T. Desell, D. Lombraña González, F. Fernández de Vega, M. Magdon-Ismail, H. Newberg, B. Szymanski, y C. Varela. Evolutionary algorithms on volunteer computing platforms: The milkyway@home project. En F. Fernández de Vega y E. Cantú-Paz, eds., *Parallel and Distributed Computational Intelligence*, tomo 269 de *Studies in Computational Intelligence*, págs. 63–90. Springer-Verlag, Berlin Heidelberg, 2010.

[67] R.J. Collins y D.R. Jefferson. Selection in massively parallel genetic algorithms. En R.K. Belew y L.B. Booker, eds., *Fourth International Conference on Genetic Algorithms*, págs. 249–256. Morgan Kaufmann Publishers, San Diego, CA, USA, 1991.

[68] S. Cook. The complexity of theorem-proving procedures. En *third annual ACM symposium on Theory of computing*, págs. 151–158. ACM, New York, NY, USA, 1971.

[69] S. Cook. An overview of computational complexity. *Communications of the ACM*, 26(6):400–408, 1983.

[70] C. Cotta. Protein structure prediction using evolutionary algorithms hybridized with backtracking. En J. Mira y J. Álvarez, eds., *Artificial Neural Nets Problem Solving Methods*, tomo 2687 de *Lecture Notes in Computer Science*, págs. 321–328. Springer-Verlag, Berlin Heidelberg, 2003.

[71] C. Cotta. Memetic algorithms with partial lamarckism for the shortest common supersequence problem. En J. Mira y J.R. Álvarez, eds., *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach*, tomo 3562 de *Lecture Notes in Computer Science*, págs. 84–91. Springer-Verlag, Berlin Heidelberg, 2005.

[72] C. Cotta, E. Alba, y J.M. Troya. Stochastic reverse hillclimbing and iterated local search. En *1999 Congress on Evolutionary Computation*, págs. 1558–1565. IEEE Press, Washington D.C., 1999.

[73] C. Cotta y A. Fernández. A hybrid GRASP - evolutionary algorithm approach to golomb ruler search. En X. Yao et al., eds., *Parallel Problem Solving From Nature VIII*, tomo 3242 de *Lecture Notes in Computer Science*, págs. 481–490. Springer-Verlag, Berlin Heidelberg, 2004.

[74] C. Cotta y P. Moscato. The k-feature set problem is W[2]-complete. *Journal of Computer and System Sciences*, 67(4):686–690, 2003.

[75] C. Cotta y F. Neri. Memetic algorithms in continuous optimization. En [263], págs. 123–136.

[76] C. Cotta, M. Sevaux, y K. Sörensen, eds. *Adaptive and Multilevel Metaheuristics*, tomo 136 de *Studies in Computational Intelligence*. Springer-Verlag, Berlin Heidelberg, 2008.

[77] C. Cotta y J.M. Troya. A hybrid genetic algorithm for the 0-1 multiple knapsack problem. En G.D. Smith et al., eds., *Artificial Neural Nets and Genetic Algorithms*, tomo 3, págs. 250–254. Springer-Verlag, Berlin Heidelberg, 1998.

[78] C. Cotta y J.M. Troya. Embedding branch and bound within evolutionary algorithms. *Applied Intelligence*, 18(2):137–153, 2003.

[79] C. Cotta y J.M. Troya. Reverse engineering of temporal boolean networks from noisy data using evolutionary algorithms. *Neurocomputing*, 62:111–129, 2004.

[80] P. Cowling, G. Kendall, y E. Soubeiga. A hyperheuristic aproach to schedule a sales submit. En E. Burke y W. Erben, eds., *Third International Conference on Practice and Theory of Automated Timetabling III - PATAT 2000*, tomo 2079 de *Lecture Notes in Computer Science*, págs. 176–190. Springer-Verlag, Berlin Heidelberg, 2000.

[81] N.L. Cramer. A representation for the adaptive generation of simple sequential programs. En J.J. Grefenstette, ed., *First International Conference on Genetic Algorithms*, págs. 183–187. Lawrence Erlbaum Associates, 1985.

[82] J. Culberson. On the futility of blind search: An algorithmic view of "no free lunch". *Evolutionary Computation*, 6(2):109–128, 1998.

[83] C. Darwin. *On the Origin of Species by Means of Natural Selection*. John Murray, London, 1859.

[84] Y. Davidor y O. Ben-Kiki. The interplay among the genetic algorithm operators: Information theory tools used in a holistic way. En [233], págs. 75–84.

[85] L Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, 1991.

[86] R. Dawkins. *The Selfish Gene*. Clarendon Press, Oxford, 1976.

[87] I. De Falco, A. Della Cioppa, D. Maisto, U. Scafuri, y E. Tarantino. Biological invasion - inspired migration in distributed evolutionary algorithms. *Information Sciences*, 207:50–65, 2012.

[88] R. de Haan y S. Szeider. Compendium of parameterized problems at higher levels of the polynomial hierarchy. *Electronic Colloquium on Computational Complexity, Report No. 143*, 2014.

[89] K. Deb y D.E. Goldberg. Analyzing deception in trap functions. En L.D. Whitley, ed., *Second Workshop on Foundations of Genetic Algorithms*, págs. 93–108. Morgan Kaufmann Publishers, Vail, Colorado, USA, 1993.

[90] A. Dekkers y E. Aarts. Global optimization and simulated annealing. *Mathematical Programming*, 50:367–393, 1991.

[91] M. den Besten, T. Stützle, y M. Dorigo. Design of iterated local search algorithms. an example application to the single machine total weighted tardiness problem. En J.W. Egbert et al., eds., *Applications of Evolutionary Computing*, tomo 2037 de *Lecture Notes in Computer Science*, págs. 441–451. Springer-Verlag, Berlin Heidelberg, 2001.

[92] J. Derrac, S. García, D. Molina, y F. Herrera. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, 1(1):3–18, 2011.

[93] E.W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, 1974.

[94] M. Dorigo y C. Blum. Ant colony optimization theory: A survey. *Theoretical Computer Science*, 344(2-3):243–278, 2005.

[95] M. Dorigo, E. Bonabeau, y G. Theraulaz. Ant algorithms and stigmergy. *Future Generation Computer Systems*, 16(8):851–871, 2000.

[96] M. Dorigo y G. Di Caro. The ant colony optimization meta-heuristic. En D. Corne, M. Dorigo, y F. Glover, eds., *New Ideas in Optimization*. McGraw-Hill, 1999.

[97] B. Dorronsoro y E. Alba. *Cellular Genetic Algorithms*, tomo 42 de *Operations Research/Computer Science Interfaces*. Springer Verlag, Berlin Heidelberg, 2008.

[98] B. Dorronsoro, D. Arias, F. Luna, A.J. Nebro, y E. Alba. A grid-based hybrid cellular genetic algorithm for very large scale instances of the CVRP. En *High Performance Computing & Simulation Conference - HPCS*, págs. 759–765. IEEE Press, Piscataway, NJ, 2007.

[99] R.G. Downey y M.R. Fellows. Fixed parameter tractability and completeness I: Basic theory. *SIAM Journal of Computing*, 24:873–921, 1995.

[100] R.G. Downey y M.R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer-Verlag, Berlin Heidelberg, 1 ed$^{\underline{\text{ón}}}$., 1999.

[101] R.G. Downey, M.R. Fellows, y K.W. Regan. Descriptive complexity and the W hierarchy. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 39:119–134, 1998.

[102] R.G. Downey, M.R. Fellows, y U. Stege. Computational tractability: The view from mars. *Bulletin of the European Association for Theoretical Computer Science*, 69:73–97, 1999.

[103] S. Droste, T. Jansen, y I.Wegener. Optimization with randomized search heuristics-the (A)NFL theorem, realistic scenarios and difficult functions. *Theoretical Computer Science*, 287(1):131–144, 2002.

[104] S. Droste, T. Jansen, y I. Wegener. Perhaps not a free lunch but at least a free appetizer. En [33], págs. 833–839.

[105] A.E. Eiben, M.C. Schut, y A.R. de Wilde. Is self-adaptation of selection pressure and population size possible? - a case study. En T.P. Runarsson et al., eds., *Parallel Problem Solving from Nature - PPSN IX*, tomo 4193 de *Lecture Notes in Computer Science*, págs. 900–909. Springer-Verlag, Berlin Heidelberg, 2006.

[106] A.E. Eiben y J.E. Smith. *Introduction to Evolutionary Computation*. Natural Computing Series. Springer-Verlag, Berlin Heidelberg, 2003.

[107] A.E. Eiben y J.E. Smith. Evolutionary algorithms. En [263], págs. 9–27.

[108] A.E. Eiben et al., eds. *Parallel Problem Solving from Nature - PPSN V*, tomo 1498 de *Lecture Notes in Computer Science*. Springer Verlag, Berlin Heidelberg, 1998.

[109] C. English, S. Terzis, y P. Nixon. Towards self-protecting ubiquitous systems: monitoring trust-based interactions. *Personal and Ubiquitous Computing*, 10(1):50–54, 2005.

[110] C. Erick, ed. *Efficient and Accurate Parallel Genetic Algorithms*. Kluwer Academic Publishers, USA, 2000.

[111] M.R. Fellows. Parameterized complexity: The main ideas and connections to practical computing. *Electronic Notes in Theoretical Computer Science*, 61, 2002.

[112] T.A. Feo y M.G.C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1999.

[113] C.M. Fernandes, J.L.J. Laredo, J.J. Merelo, C. Cotta, R. Nogueras, y A.C. Rosa. Shuffle and mate: A dynamic model for spatially structured evolutionary algorithms. En [37], págs. 50–59.

[114] C.M. Fernandes, J.L.J. Laredo, J.J. Merelo, C. Cotta, y A. Rosa. Towards a 2-dimensional self-organized framework for structured population-based metaheuristics. En *International Conference on Complex Systems*, págs. 1–6. IEEE Press, 2012.

[115] C.M. Fernandes, J.L.J. Laredo, J.J. Merelo, C. Cotta, y A.C. Rosa. Dynamic and partially connected ring topologies for evolutionary algorithms with structured populations. En A.I. Esparcia-Alcázar y A. Mora, eds., *Applications of Evolutionary Computation*, tomo 8602 de *Lecture Notes in Computer Science*, págs. 665–677. Springer-Verlag, Berlin Heidelberg, 2014.

[116] C.M. Fernandes, J.L.J. Laredo, A.C. Rosa, y J.J. Merelo. The sandpile mutation genetic algorithm: an investigation on the working mechanisms of a diversity-oriented and self-organized mutation operator for non-stationary functions. *Applied Intelligence*, 39(2):279–306, 2013.

[117] C.M. Fernandes, J.J. Merelo, y A.C. Rosa. Controlling the parameters of the particle swarm optimization with a self-organized criticality model. En C.A. Coello Coello et al., eds., *Parallel Problem Solving from Nature - PPSN XII*, tomo 7492 de *Lecture Notes in Computer Science*, págs. 153–163. Springer-Verlag, Berlin Heidelberg, 2012.

[118] C.M. Fernandes y A.C. Rosa. Self-adjusting the intensity of assortative mating in genetic algorithms. *Soft Computing*, 12(10):955–979, 2007.

[119] C.M. Fernandes, A.C. Rosa, J.L.J. Laredo, C. Cotta, y J.J. Merelo. A study on time-varying partially connected topologies for the particle swarm. En *2013 IEEE Congress on Evolutionary Computation - CEC*, págs. 2450–2456. IEEE Press, 2013.

[120] F. Fernández, L. Vanneschi, y M. Tomassini. The effect of plagues in genetic programming: A study of variable-size populations. En C. Ryan et al., eds., *Genetic Programming*, tomo 2610 de *Lecture Notes in Computer Science*, págs. 317–326. Springer-Verlag, Berlin Heidelberg, 2003.

[121] F. Fernández de Vega, E. Cantú-Paz, J.I. López, y T. Manzano. Saving resources with plagues in genetic algorithms. En X. Yao et al., eds., *Parallel Problem Solving from Nature - PPSN VIII*, tomo 3242 de *Lecture Notes in Computer Science*, págs. 272–281. Springer-Verlag, Berlin Heidelberg, 2004.

[122] J. Flum y M. Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin Heidelberg, 1 ed$^{\underline{ón}}$., 2006.

[123] M.J. Flynn. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, 21:948–960, 1972.

[124] D.B. Fogel. *Evolutionary Computation: The Fossil Record*. Wiley-IEEE Press, Piscataway, NJ, 1998.

[125] G.B. Fogel y D.W. Corne. *Evolutionary Computation in Bioinformatics*. Morgan Kaufmann Publishers, 2003.

[126] L.J. Fogel, A.J. Owens, y M.J. Walsh. *Artificial Intelligence Through Simulated Evolution*. Wiley, New York, 1966.

[127] G. Fox. Introduction to web computing. *Computing in Science & Engineering*, 3:52–53, 2001.

[128] B. Freisleben y P. Merz. A genetic local search algorithm for solving symmetric and asymmetric traveling salesman problems. En *1996 IEEE International Conference on Evolutionary Computation*, págs. 616–621. IEEE Press, Nagoya, Japan, 1996.

[129] M. Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association (American Statistical Association)*, 32(200):675–701, 1937.

[130] M. Friedman. A correction: The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association (American Statistical Association)*, 34(205):109, 1939.

[131] M. Friedman. A comparison of alternative tests of significance for the problem of m rankings. *The Annals of Mathematical Statistics*, 11(1):86–92, 1940.

[132] W. Furmanski. Petaops and exaops: Super-computing on the web. *IEEE Internet Computing*, 1(2):38–46, 1997.

[133] L.M. Gabora. Meme and variations: A computational model of cultural evolution. En L. Nadel y D.L. Stein, eds., *Lectures in Complex Systems*, págs. 471–494. Addison-Wesley, 1993.

[134] C. Gagné, M. Parizeau, y M. Dubreuil. Distributed beagle: An environment for parallel and distributed evolutionary computations. En *17th Annual International Symposium on High Performance Computing Systems and Applications - HPCS 2003*, págs. 201–208. Sherbrooke, Quebec, Canada, 2013.

[135] M. García Arenas, P. Collet, A.E. Eiben, M. Jelasity, J.J. Merelo Guervós, B. Paechter, M. Preuß, y M. Schoenauer. A framework for distributed evolutionary algorithms. En [243], págs. 665–675.

[136] P. García-Sánchez, M.I. García Arenas, A.M. Mora, P.A. Castillo, C. Fernandes, P. de las Cuevas, G. Romero, J. González, y J.J. Merelo. Developing services in a service oriented architecture for evolutionary algorithms. En *Genetic and Evolutionary Computation - GECCO 2013*, págs. 1341–1348. ACM Press, New York, NY, USA, 2013.

[137] P. García-Sánchez, J. González, P.A. Castillo, M. García Arenas, y J.J. Merelo Guervós. Service oriented evolutionary algorithms. *Soft Computing*, 17(6):1059–1075, 2013.

[138] M. García-Valdez, L. Trujillo, F. Fernández de Vega, J.J Merelo Guervós, y G. Olague. Evospace: A distributed evolutionary platform based on the tuple space model. En A.I. Esparcia-Alcázar, ed., *Applications of Evolutionary Computation*, tomo 7835 de *Lecture Notes in Computer Science*, págs. 499–508. Springer Verlag, Berlin Heidelberg, 2013.

[139] M. García-Valdez, L. Trujillo, F. Fernández de Vega, J.J Merelo Guervós, y G. Olague. Evospace-interactive: A framework to develop distributed collaborative-interactive evolutionary algorithms for artistic design. En P. Machado, J. McDermott, y A. Carballal, eds., *Evolutionary and Biologically Inspired Music, Sound, Art and Design*, tomo 7834 de *Lecture Notes in Computer Science*, págs. 121–132. Springer Verlag, Berlin Heidelberg, 2013.

[140] M. García-Valdez, L. Trujillo, J.J. Merelo-Guervós, y F Fernández de Vega. Randomized parameter settings for heterogeneous workers in a pool-based evolutionary algorithm. En [37], págs. 702–710.

[141] M.R. Garey y D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Series of Books in the Mathematical Sciences. W. H. Freeman, 1 edón., 1979.

[142] M. Gendreau y J.Y. Potvin, eds. *Handbook of Metaheuristics*, tomo 146 de *International Series in Operations Research and Management Science*. Springer-Verlag, Berlin Heidelberg, 2 edón., 2010.

[143] I.P. Gent y T. Walsh. The SAT phase transition. En A.G. Cohn, ed., *Proceedings of 11th European Conference on Artificial Intelligence*, págs. 105–109. John Wiley & Sons, 1994.

[144] D. Ghosh, R. Sharman, H.R. Rao, y S. Upadhyaya. Self-healing systems - survey and synthesis. *Decision Support Systems*, 42(4):2164–2185, 2007.

[145] M. Giacobini, E. Alba, A. Tettamanzi, y M. Tomassini. Selection intensity in cellular evolutionary algorithms for regular lattices. *IEEE Transactions on Evolutionary Computation*, 9(5):489–505, 2005.

[146] M. Giacobini, E. Alba, y M. Tomassini. Selection intensity in asynchronous cellular evolutionary algorithms. En E. Cantú-Paz et al., eds., *Genetic and Evolutionary Computation - GECCO 2003*, tomo 2723 de *Lecture Notes in Computer Science*, págs. 955–966. Springer Verlag, Berlin Heidelberg, 2003.

[147] M. Giacobini, M. Tomassini, y A. Tettamanzi. Modelling selection intensity for linear cellular evolutionary algorithms. En P. Liardet et al., eds., *Artificial Evolution*, tomo 2936 de *Lecture Notes in Computer Science*, págs. 345–356. Springer Verlag, Berlin Heidelberg, 2003.

[148] D. Gil, D. Roche, A. Borràs, y J. Giraldo. Terminating evolutionary algorithms at their steady state. *Computational Optimization and Applications*, 61(2):489–515, 2015.

[149] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13:533–549, 1986.

[150] F. Glover, J.P. Kelly, y M. Laguna. Genetic algorithms and tabu search: Hybrids for optimization. *Computers & Operations Research*, 22(1):111–134, 1995.

[151] F. Glover y G. Kochenberger, eds. *Handbook of Metaheuristics*, tomo 57 de *International Series in Operations Research and Management Science*. Kluwer Academics Publishers, Boston MA, 1 ed$^{\underline{\text{ón}}}$., 2003.

[152] F. Glover y M. Laguna. *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, 1997.

[153] D.E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison-Wesley Co., 1989.

[154] D.E. Goldberg, K. Deb, y J. Horn. Massive multimodality, deception and genetic algorithms. En [233], págs. 37–48.

[155] O. Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 1 ed. ed$^{\underline{\text{ón}}}$., 2008.

[156] C. González, J.A. Lozano, y P. Larrañaga. Mathematical modeling of discrete estimation of distribution algorithms. En P. Larrañaga y J.A. Lozano, eds., *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*, tomo 2 de *Genetic Algorithms and Evolutionary Computation*, págs. 147–163. Springer Verlag, Berlin Heidelberg, 2002.

[157] D.L. González, F. Fernández de Vega, L. Trujillo, G. Olague, L. Araujo, P. Castillo, J.J. Merelo, y K. Sharman. Increasing GP computing power for free via desktop GRID computing and virtualization. En *Parallel, Distributed and Network-based Processing*, págs. 419–423. IEEE Press, 2009.

[158] J.A. González, C. Rodríguez, G. Rodríguez, F. de Sande, y M. Printista. A tool for performance modeling of parallel programs. *Scientific Computing*, 11(3):191–198, 2003.

[159] T.F. González, ed. *Handbook of Approximation Algorithms and Metaheuristics*. Chapman & Hall/Crc Computer & Information Science Series, 2007.

[160] M. Gorges-Schleuter. ASPARAGOS: an asynchronous parallel genetic optimization strategy. En J.D. Schaffer, ed., *Third International Conference on Genetic Algorithms*, págs. 422–427. Morgan Kaufmann Publishers, San Francisco, CA, 1989.

[161] S.J. Gould y N. Elredge. Punctuated equilibria: The tempo and mode of evolution reconsidered. *Paleobiology*, 32:115–151, 1977.

[162] J. Grefenstette. Genetic algorithms for changing environments. En [233], págs. 137–144.

[163] J.J. Grefenstette. Parallel adaptive algorithms for function optimization. Inf. Téc. CS-81-19, Vanderbilt University, Nashville, TN, 1981.

[164] J.J. Grefenstette. Virtual genetic algorithms: First results. Inf. Téc. AIC-95-013, Navy Center for Applied Research in Artificial Intelligence, 1995.

[165] P.B. Grosso. *Computer simulation of genetic adaptation: Parallel subcomponent interaction in a multilocus model*. Tesis Doctoral, University of Michigan, Ann Arbor, 1985.

[166] A. Grushin. The effects of static fitness function noise upon the performance of genetic algorithms. En H.D. Cheng, S.D. Chen, y R.Y. Lin, eds., *7th Joint Conference on Information Sciences - JCIS 2006*, págs. 275–278. Atlantis Press, Kaohsiung, Taiwan, 2006.

[167] Q. Guo, T. Zhou, J.G. Liu, Q.J. Bai, B.H. Wang, y M. Zhao. Growing scale-free small-world networks with tunable assortative coefficient. *Physica A: Statistical Mechanics and its Applications*, 371:814–822, 2006.

[168] M.T. Hallett y H.T. Wareham. A compendium of parameterized complexity results. *ACM SIGACT News*, 25(3):122–123, 1994. New York, NY, USA.

[169] W.E. Hart y R.K. Belew. Optimizing an arbitrary function is hard for the genetic algorithm. En R.K. Belew y L.B. Booker, eds., *Fourth International Conference on Genetic Algorithms*, págs. 190–195. Morgan Kaufmann Publishers, San Mateo CA, 1991.

[170] W.E. Hart, N. Krasnogor, y J.E. Smith. *Recent Advances in Memetic Algorithms*, tomo 166 de *Studies in Fuzziness and Soft Computing*. Springer-Verlag, Berlin Heidelberg, 2005.

[171] H. Hermes. *Aufzählbarkeit, Entscheidbarkeit, Berechenbarkeit*. Springer-Verlag, Berlin Heidelberg, 1971.

[172] J.I. Hidalgo, J. Lanchares, F Fernández de Vega, y D. Lombraña. Is the island model fault tolerant? En [347], págs. 2737 – 2744.

[173] R. Hinterding, Z. Michalewicz, y A.E. Eiben. Adaptation in evolutionary computation: A survey. En *Fourth IEEE Conference on Evolutionary Computation*, págs. 65–69. IEEE Press, Piscataway, New Jersey, 1997.

[174] J.H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Harbor, 1975.

[175] S. Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6:65–70, 1979.

[176] P. Holme y B.J. Kim. Growing scale-free networks with tunable clustering. *Physics Review E*, 65(026107), 2002.

[177] H. Hoos y T. Stützle. *Stochastic Local Search - Foundations and Applications*. Morgan Kaufmann Publishers, 2005.

[178] J.E. Hopcroft y J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Series in Computer Science. Addison-Wesley, 1993.

[179] R.M. Hord. *Understanding Parallel Supercomputing*. Understanding Science and Technology Series. IEEE Press, 1999.

[180] P. Horn. Autonomic computing: IBM's perspective on the state of information technology. Inf. téc., IBM Research, 2001. URL http://people.scs.carleton.ca/~soma/biosec/readings/autonomic_computing.pdf. Accessed 14/05/2015.

[181] C.R. Houck, J.A. Joines, M.G. Kay, y J.R. Wilson. Empirical investigation of the benefits of partial lamarckianism. *Evolutionary Computation*, 5(1):31–60, 1997.

[182] M.C. Huebscher y J.A. McCann. A survey of autonomic computing - degrees, models and applications. *ACM Computing Surveys*, 40(3), 2008. Article 7.

[183] M. Hulin. An optimal stop criterion for genetic algorithms: A bayesian approach. En T. Bäck, ed., *Seventh International Conference on Genetic Algorithms*, págs. 135–143. Morgan Kaufmann Publishers, San Mateo, CA, 1997.

[184] J. Huxley. *Evolution, the Modern Synthesis*. Harper, New York NY, 1942.

[185] C. Igel y M. Toussaint. On classes of functions for which no free lunch results hold. *Information Processing Letters*, 86(6):317–321, 2003.

[186] J.A J.A. Lozano, P. Larrañaga, I. Inza, y E. Bengoetxea, eds. *Towards a New Evolutionary Computation*, tomo 192 de *Studies in Fuzziness and Soft Computing*. Springer Verlag, Berlin Heidelberg, 2006.

[187] M. Jelasity y M. van Steen. Large-scale newscast computing on the internet. Inf. Téc. IR-503, Vrije Universiteit Amsterdam, Department of Computer Science, Amsterdam, The Netherlands, 2002.

[188] T.C. Jones. *Fitness Landscapes and Search*. Tesis Doctoral, University of New Mexico, 1995.

[189] I. Karcz-Duleba. Time to the convergence of evolution in the space of population states. *International Journal of Applied Mathematics and Computer Science*, 14(3):279–287, 2004.

[190] A.H. Karp y H.P Flatt. Measuring parallel processor performance. *Communications of the ACM*, 33(5):539–543, 1990.

[191] G. Kendall, P. Cowling, y E. Sou. Choice function and random hyperheuristics. En L. Wang et al., eds., *Fourth Asia-Pacific Conference on Simulated Evolution and Learning*, págs. 667–671. 2002.

[192] J. Kennedy y R. Eberhart. Particle swarm optimization. En *International Conference of Neural Networks - ICNN 1995*, tomo 4, págs. 1942–1948. IEEE Press, Perth, Australia, 1995.

[193] J. Kennedy y R.C. Eberhart, eds. *Swarm Intelligence*. Morgan Kaufmann Publishers, San Francisco, CA, USA, 2001.

[194] J.O. Kephart y D.M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[195] M. Kimura. Evolutionary rate at the molecular level. *Nature*, 217:624–626, 1968.

[196] P.N. Klein y N.E. Young. Algorithms and theory of computation handbook. En M. J. Atallah y M. Blanton, eds., *Approximation Algorithms for NP-Hard Optimization Problems*, tomo 1, cap. 34, págs. 1–23. Chapman & Hall/CRC, 2010.

[197] A.V. Kononova, D.B. Ingham, y M. Pourkashanian. Simple scheduled memetic algorithm for inverse problems in higher dimensions: Application to chemical kinetics. En *IEEE Congress on Evolutionary Computation 2008*, págs. 3906–3913. IEEE Press, Hong Kong, 2008.

[198] J.R. Koza. *Genetic Programming*. MIT Press, Cambridge MA, 1992.

[199] S. Koziel y Z. Michalewicz. A decoder-based evolutionary algorithm for constrained parameter optimization problems. En [108], págs. 231–240.

[200] N. Krasnogor. *Studies in the Theory and Design Space of Memetic Algorithms*. Tesis Doctoral, University of the West of England, 2002.

[201] N. Krasnogor. Self generating metaheuristics in bioinformatics: The proteins structure comparison case. *Genetic Programming and Evolvable Machines*, 5(2):181–201, 2004.

[202] N. Krasnogor, B. Blackburne, E. Burke, y J. Hirst. Multimeme algorithms for protein structure prediction. En [243], págs. 769–778.

[203] N. Krasnogor y S. Gustafson. Toward truly "memetic"memetic algorithms: discussion and proof of concepts. En D. Corne et al., eds., *Advances in Natural -Inspired Computation - PPSN VII*, págs. 9–10. University of Reading, Reading, UK, 2002.

[204] N. Krasnogor y S. Gustafson. A study on the use of "self-generation" in memetic algorithms. *Natural Computing*, 3(1):53–76, 2004.

[205] N. Krasnogor y J.E. Smith. A memetic algorithm with self-adaptive local search: TSP as a case study. En [364], págs. 987–994.

[206] N. Krasnogor y J.E. Smith. Emergence of profitable search strategies based on a simple inheritance mechanism. En [336], págs. 432–439.

[207] N. Krasnogor y J.E. Smith. A tutorial for competent memetic algorithms: Model, taxonomy and desing issues. *IEEE Transactions on Evolutionary Computation*, 9:474–488, 2005.

[208] N. Krasnogor y J.E. Smith. Memetic algorithms: The polynomial local search complexity theory perspective. *Journal of Mathematical Modelling and Algorithms*, 7(1):3–24, 2008.

[209] V. Kumar, A. Grama, A. Gupta, y G. Karypis. *Introduction to Parallel Computing Design and Analysis of Algorithms*. The benjamin / Cummings Publishing Company Inc., 1994.

[210] J.B. Lamarck. *Philosophie Zoologique*. Museum d'Histoire Naturelle (Jardin des Plantes), 1809.

[211] C.G. Langton. Artificial life. *Artificial Life 1*, págs. 1–47, 1989.

[212] J.L.J. Laredo, P. Bouvry, D.L. González, F. Fernández de Vega, M.G. Arenas, J.J. Merelo, y C.M. Fernandes. Designing robust volunteer-based evolutionary algorithms. *Genetic Programming and Evolvable Machines*, 15(3):221–244, 2014.

[213] J.L.J. Laredo, P.A. Castillo, A.M. Mora, y J.J. Merelo. Evolvable agents, a fine grained approach for distributed evolutionary computing: walking towards the peer-to-peer computing frontiers. *Soft Computing*, 12(12):1145–1156, 2008.

[214] J.L.J. Laredo, P.A. Castillo, A.M. Mora, J.J. Merelo, y C. Fernandes. Resilience to churn of a peer-to-peer evolutionary algorithm. *International Journal of High Performance Systems Architecture*, 1(4):260–268, 2008.

[215] J.L.J. Laredo, A. E. Eiben, M. van Steen, y J.J. Merelo. Evag: a scalable peer-to-peer evolutionary algorithm. *Genetic Programming and Evolvable Machines*, 11(2):227–246, 2010.

[216] J.L.J. Laredo, J.J. Merelo Guervós, y P. Castillo Valdivieso. Evolvable agents: A framework for peer-to-peer evolutionary algorithms. En F. Fernández de Vega y E. Cantú-Paz, eds., *Parallel and Distributed Computational Intelligence*, tomo 269 de *Studies in Computational Intelligence*, págs. 43–62. Springer-Verlag, Berlin Heidelberg, 2010.

[217] J.L.J. Laredo, D. Lombraña González, F. Fernández de Vega, M. García Arenas, y J.J. Merelo Guervós. A peer-to-peer approach to genetic programming. En S. Silva et al., eds., *Genetic Programming*, tomo 6621 de *Lecture Notes in Computer Science*, págs. 108–117. Springer-Verlag, Berlin Heidelberg, 2011.

[218] P. Larrañaga y J.A. Lozano, eds. *Estimation of Distribution Algorithms*, tomo 2 de *Genetic Algorithms and Evolutionary Computation*. Springer Verlag, Berlin Heidelberg, 2002.

[219] J. Lässig y D. Sudholt. The benefit of migration in parallel evolutionary algorithms. En [295], págs. 1105–1112.

[220] M.N. Le, Y.S. Ong, Y. Jin, y B. Sendhoff. Lamarckian memetic algorithms: Local optimum and connectivity structure analysis. *Memetic Computing Journal*, 1(3):175–190, 2009.

[221] C. A. Lee, C. DeMatteis, J. Stepanek, y J. Wang. Cluster performance and the implications for distributed, heterogeneous grid performance. En *Heterogeneous Computing Workshop*, págs. 253–261. 2000.

[222] E.T. Lee y J.W. Wang, eds. *Statistical Methods for Survival Data Analysis*. John Wiley & Sons, Hoboken, NJ, USA, 2003.

[223] A. Legrand, H. Renard, Y. Robert, y F. Vivien. Load-balancing iterative computations in heterogenous clusters with shared communication links. En R. Wyrzykowski et al., eds., *Parallel Processing and Applied Mathematics*, tomo 3019 de *Lecture Notes in Computer Science*, págs. 930–937. 2004.

[224] S. Lin y B. Kernighan. An effective heuristic algorithm for the traveling salesman. *Operations Research*, 21:498–516, 1973.

[225] C. Liu, R.W. White, y S. Dumais. Understanding web browsing behaviors through weibull analysis of dwell time. En *33rd International ACM SIGIR Conference on Research and Development in Information Retrieval - SIGIR 2010*, págs. 379–386. ACM Press, New York, NY, USA, 2010.

[226] D. Lombraña González, F. Fernández de Vega, y H. Casanova. Characterizing fault tolerance in genetic programming. *Future Generation Computer Systems*, 26(6):847–856, 2010.

[227] D. Lombraña González, J.L. Jiménez Laredo, F. Fernández de Vega, y J.J. Merelo Guervós. Characterizing fault-tolerance of genetic algorithms in desktop grid systems. En P. Cowling y P. Merz, eds., *Evolutionary Computation in Combinatorial Optimization*, tomo 6022 de *Lecture Notes in Computer Science*, págs. 131–142. Springer-Verlag, Berlin Heidelberg, 2010.

[228] D. Lombraña González, J.L. Jiménez Laredo, F. Fernández de Vega, y J.J. Merelo Guervós. Characterizing fault-tolerance in evolutionary algorithms. En F. Fernández de Vega et al., eds., *Parallel Architectures and Bioinspired Algorithms*, tomo 415 de *Studies in Computational Intelligence*, págs. 77–99. Springer-Verlag, Berlin Heidelberg, 2012.

[229] H.R. Lourenço, O. Martin, y T. Stützle. Iterated local search. En [142], págs. 321–353.

[230] R. Lüling, B. Monien, y F. Ramme. Load balancing in large networks: a comparative study. En *Third IEEE Symposium on Parallel and Distributed Processing*, págs. 686–689. IEEE Press, 1991.

[231] G. Luque y E. Alba. Selection pressure and takeover time of distributed evolutionary algorithms. En [295], págs. 1083–1088.

[232] A. Mambrini y D. Sudholt. Design and analysis of adaptive migration intervals in parallel evolutionary algorithms. En C. Igel, ed., *Genetic and Evolutionary Computation - GECCO 2014*, págs. 1047–1054. ACM Press, New York, NY, USA, 2014.

[233] R. Männer y B. Manderick, eds. *Parallel Problem Solving from Nature - PPSN II*. Elsevier Science Inc., New York, NY, USA, 1992.

[234] S. Margetts. *Adaptive Genotype to Phenotype Mappings for Evolutionary Algorithms*. Tesis Doctoral, Department of Computer Science, Cardiff University, 2001.

[235] T. Maruyama, T. Hirose, y A. Konagaya. A fine-grained parallel genetic algorithm for distributed parallel systems. En S. Forrest, ed., *5th International Conference on Genetic Algorithms*, págs. 184–190. 1993.

[236] A.J. McMurry, C.A. Gilbert, B.Y. Reis, H.C. Chueh, I.S. Kohane, y K.D. Mandl. A self-scaling, distributed information architecture for public health, research and clinical care. *Journal of the American Medical Informatics Association*, 14(4):527–533, 2007.

[237] N. Melab, S. Cahon, y E.G. Talbi. Grid computing for parallel bioinspired algorithms. *Journal of Parallel and Distributed Computing*, 66(8):1052–1061, 2006.

[238] N. Melab, M. Mezmaz, y E.G. Talbi. Parallel hybrid multi-objective island model in peer-to-peer environment. En *19th IEEE International Parallel and Distributed Processing Symposium - IPDPS 2005 - Workshop 6*, tomo 7, pág. 190b. IEEE Computer Society, Washington, DC, USA, 2005.

[239] G. Mendel. Versuche über Pflanzen-Hybriden. *Verhandlungen des Naturforschendes Vereines in Brünn*, 4:3–47, 1865.

[240] A. Mendes, C. Cotta, V. García, P. França, y P. Moscato. Gene ordering in microarray data using parallel memetic algorithms. En T. Skie y C.S. Yang, eds., *International Conference on Parallel Processing Workshops*, págs. 604–611. IEEE Press, Oslo, Norway, 2005.

[241] J.J. Merelo Guervós, M. García Arenas, A.M. Mora, P.A. Castillo, G. Romero, y J.L.J. Laredo. Cloud-based evolutionary algorithms: An algorithmic study. *Computing Research Repository - CoRR*, abs/1105.6205, 2011.

[242] J.J. Merelo Guervós, A. Mora García, J.L.J. Laredo, J. Lupión, y F. Tricas. Browser-based distributed evolutionary computation: performance and scaling behavior. En [347], págs. 2851–2858.

[243] J.J. Merelo Guervós et al., eds. *Parallel Problem Solving from Nature - PPSN VII*, tomo 2439 de *Lecture Notes in Computer Science*. Springer Verlag, Berlin Heidelberg, 2002.

[244] K. Meri, M.G. Arenas, A.M. Mora, J.J. Merelo, P.A. Castillo, P. García-Sánchez, y J.L.J. Laredo. Cloud-based evolutionary algorithms: An algorithmic study. *Natural Computing*, 12(2):135–147, 2013.

[245] P. Merz y A. Zell. Clustering gene expression profiles with memetic algorithms. En [243], págs. 811–820.

[246] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, Berlin Heidelberg, 1992.

[247] Z. Michalewicz. A survey of constraint handling techniques in evolutionary computation methods. En J.R. McDonnell et al., eds., *Evolutionary Programming IV*, págs. 135–155. MIT Press, 1995.

[248] Z. Michalewicz. Repair algorithms. En [29], págs. C5.4:1–5.

[249] M.J. Mihaljević y H. Imai. Security issues of cloud computing and an encryption approach. En M. Despotović-Zrakić, M. Milutinović, y A. Belić, eds., *Handbook of Research on High Performance and Cloud Computing in Scientific Research and Education*, págs. 388–408. IGI Global, Hershey, PA, USA, 2014.

[250] D. Millán-Ruiz y J.I. Hidalgo. Migration and replacement policies for preserving diversity in dynamic environments. En C. Di Chio et al., eds., *Applications of Evolutionary Computation*, tomo 7248 de *Lecture Notes in Computer Science*, págs. 456–465. Springer-Verlag, Berin Heidelberg, 2012.

[251] D.S. Milojičić, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, y Z. Xu. Peer-to-peer computing. Inf. Téc. HPL-2002-57, Hewlett-Packard Labs, 2002.

[252] N. Mladenović y P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24:1097–1100, 1997.

[253] R.L.V. Moritz y M. Middendorf. Decentralized and dynamic group formation of reconfigurable agents. *Memetic Computing*, 7(2):77–91, 2015.

[254] P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Inf. Téc. 826, Technical Report Caltech Concurrent Computation Program, California Institute of Technology, Pasadena, California, USA, 1989.

[255] P. Moscato. Memetic algorithms: A short introduction. En D. Corne, M. Dorigo, y F. Glover, eds., *New Ideas in Optimization*, págs. 219–234. McGraw-Hill, Maidenhead, Berkshire, England, UK, 1999.

[256] P. Moscato y C. Cotta. A gentle introduction to memetic algorithms. En [151], págs. 105–144.

[257] P. Moscato y C. Cotta. A modern introduction to memetic algorithms. En [142], págs. 141–183.

[258] P. Moscato, A. Mendes, y C. Cotta. Memetic algorithms. En G. Onwubolu y B. Babu, eds., *New Optimization Techniques in Engineering*, págs. 53–85. Springer-Verlag, Berlin Heidelberg, 2004.

[259] H. Mühlenbein y G. Paaß. From recombination of genes to the estimation of distributions I. Binary parameters. En H.M. Voigt, W. Ebeling, I. Rechenberg, y H.P. Schwefel, eds., *Parallel Problem Solving from Nature - PPSN IV*, tomo 1141 de *Lecture Notes in Computer Science*, págs. 178–187. Springer-Verlag, Berlin Heidelberg, 1996.

[260] F. Neri. Diversity management in memetic algorithms. En [263], págs. 155–167.

[261] F. Neri y C. Cotta. Memetic algorithms and memetic computing optimization: A literature review. *Swarm and Evolutionary Computation*, 2:1–14, 2012.

[262] F. Neri y C. Cotta. A primer on memetic algorithms. En [263], págs. 43–52.

[263] F. Neri, C. Cotta, y P. Moscato, eds. *Handbook of Memetic Algorithms*, tomo 379 de *Studies in Computational Intelligence*. Springer-Verlag, Berlin Heidelberg, 2012.

[264] F. Neri, V. Tirronen, T. Kärkkäinen, y T. Rossi. Fitness diversity based adaptation in multimeme algorithms: A comparative study. En *IEEE Congress on Evolutionary Computation - CEC 2007*, págs. 2374–2381. IEEE Press, Singapore, 2007.

[265] Q.C. Nguyen, Y.S. Ong, y M.H. Lim. A probabilistic memetic framework. *IEEE Transactions on Evolutionary Computation*, 13(3):604–623, 2009.

[266] R. Nogueras y C. Cotta. Using statistical techniques to predict GA performance. En J. Mira y A. Prieto, eds., *Connectionist Models of Neurons, Learning Processes, and Artificial Intelligence*, tomo 2084 de *Lecture Notes in Computer Science*, págs. 474–481. Springer-Verlag, Berlin Heidelberg, 2001.

[267] R. Nogueras y C. Cotta. On partitioned fitness distributions of genetic operators for predicting GA performance. En R. Roy et al., eds., *Soft Computing in Industry - Recent Applications*, págs. 729–739. Springer-Verlag, Berlin Heidelberg, 2002.

[268] R. Nogueras y C. Cotta. Using distribution-based operators in genetic algorithms. En *Proceedings of the 6th Metaheuristic International Conference*, págs. 707–712. Universität Wien, Vienna, Austria, 2005.

[269] R. Nogueras y C. Cotta. Analyzing meme propagation in multimemetic algorithms: Initial investigations. En *Federated Conference on Computer Science and Information Systems*, págs. 1013–1019. IEEE Press, Cracow, Poland, 2013.

[270] R. Nogueras y C. Cotta. An analysis of migration strategies in island-based multimemetic algorithms. En [37], págs. 731–740.

[271] R. Nogueras y C. Cotta. On meme self-adaptation in spatially-structured multimemetic algorithms. En Ivan Dimov, Stefka Fidanova, y Ivan Lirkov, eds., *Numerical Methods and Applications - 8th International Conference*, tomo 8962 de *Lecture Notes in Computer Science*, págs. 70–77. Springer Verlag, Berlin Heidelberg, 2014.

[272] R. Nogueras y C. Cotta. A study on multimemetic estimation of distribution algorithms. En [37], págs. 322–331.

[273] R. Nogueras y C. Cotta. Self-balancing multimemetic algorithms in dynamic scale-free networks. En A.M. Mora y G. Squillero, eds., *Applications of Evolutionary Computing*, tomo 9028 de *Lecture Notes in Computer Science*, págs. 177–188. Springer Verlag, Berlin Heidelberg, 2015.

[274] R. Nogueras y C. Cotta. Self-sampling strategies for multimemetic algorithms in unstable computational environments. En J.M. Ferrández Vicente et al., eds., *Bioinspired Computation in Artificial Systems*, tomo 9108 de *Lecture Notes in Computer Science*, págs. 69–78. Springer Verlag, Berlin Heidelberg, 2015.

[275] R. Nogueras y C. Cotta. Sensitivity analysis of checkpointing strategies for multimemetic algorithms on dynamic complex networks. En *10th International Conference on Large Scale Scientific Computations*, Lecture Notes in Computer Science. Springer Verlag, Berlin Heidelberg, 2015.

[276] R. Nogueras y C. Cotta. A study on meme propagation in multimemetic algorithms. *International Journal of Applied Mathematics and Computer Science*, 25(3), 2015. In Press.

[277] R. Nogueras y C. Cotta. Studying fault-tolerance in island-based evolutionary and multimemetic algorithms. *Journal of Grid Computing*, 2015. doi:10.1007/s10723-014-9315-6. Online.

[278] R. Nogueras y C. Cotta. Studying self-balancing strategies in island-based multimemetic algorithms. *Journal of Computational and Applied Mathematics*, 2015. doi:10.1016/j.cam.2015.03.047. Online.

[279] R. Nogueras y C. Cotta. Towards resilient multimemetic systems on unstable networks with complex topology. En G. Papa, ed., *Advances in Evolutionary Algorithm Research*. Nova Science Pub., 2015. In Press.

[280] R. Nogueras, C. Cotta, C.M. Fernandes, J.L. Jiménez Laredo, J.J. Merelo, y A.C. Rosa. An analysis of a selecto-lamarckian model of multimemetic algorithms with dynamic self-organized topology. En A.H. Dediu et al., eds., *Theory and Practice of Natural Computing*, tomo 8273 de *Lecture Notes in Computer Science*, págs. 205–216. Springer-Verlag, Berlin Heidelberg, 2013.

[281] M.G. Norman y P. Moscato. A competitive and cooperative approach to complex combinatorial search. En *20th Informatics and Operations Research Meeting*, págs. 3.15–3.29. Buenos Aires, 1989.

[282] G. Ochoa, C. Mädler-Kron, R. Rodríguez, y K. Jaffe. Assortative mating in genetic algorithms for dynamic problems. En F. Rothlauf et al., eds., *Applications of Evolutionary Computing*, tomo 3449 de *Lecture Notes in Computer Science*, págs. 617–622. 2005.

[283] S. Olariu y A.Y. Zomaya, eds. *Handbook Of Bioinspired Algorithms And Applications*. Chapman & Hall/Crc Computer & Information Science, Boca Ratón FL, 2005.

[284] Y.S. Ong y A.J. Keane. Meta-lamarckian learning in memetic algorithm. *IEEE Transactions on Evolutionary Computation*, 8(2):99–110, 2004.

[285] Y.S. Ong, M.H. Lim, y X. Chen. Memetic computation –past, present and future. *IEEE Computational Intelligence Magazine*, 5(2):24–31, 2010.

[286] Y.S. Ong, M.H. Lim, Z. Zhu, y K.W. Wong. Classification of adaptive memetic algorithms: a comparative study. *IEEE Transactions on Systems Man and Cybernetics, Part B*, 36(1):141–152, 2006.

[287] I.H. Osman y G. Laporte. Metaheuristics: A bibliography. *Annals of Operations Research*, 41:421–451, 1996.

[288] K. Osorio, E. Alba, y G. Luque. Using theory to self-tune migration periods in distributed genetic algorithms. En *IEEE Congress on Evolutionary Computation*, págs. 2595–2601. IEEE Press, 2013.

[289] I.C.A. Oyeka y G.U. Ebuh. Modified wilcoxon signed-rank test. *Open Journal of Statistics*, 2:172–176, 2012.

[290] M. Ozawa. Halting of quantum turing machines. En *Unconventional Models of Computation*, tomo 2509 de *Lecture Notes in Computer Science*, págs. 58–65. Springer-Verlag, Berlin Heidelberg, 2002.

[291] C. Papadimitriou y M. Yannakakis. On limited nondeterminism and the complexity of vc dimension. *Journal of Computer and System Sciences*, 53:161–170, 1996.

[292] C.H. Papadimitriou. Computational complexity. En *Encyclopedia of Computer Science*, págs. 260–265. John Wiley and Sons Ltd., Chichester, UK, 2003.

[293] C.H. Papadimitriou y K. Steiglitz. *Combinatorial Optimization–Algorithms and Complexity*. Dover Publications, 1982.

[294] L. Pastor y J.L. Bosque. An eficiency and scalability model for heterogeneous clusters. En *IEEE International Conference on Cluster Computing - CLUSTER 2001*, págs. 427–434. 2001.

[295] M. Pelikan y J. Branke, eds. *Genetic and Evolutionary Computation - GECCO 2010*. ACM Press, New York, NY, USA, 2010.

[296] M. Pelikan y D. Goldberg. Hierarchical BOA solves ising spin glasses and MAXSAT. En E. Cantú-Paz et al., eds., *Genetic and Evolutionary Computation*, tomo 2724 de *Lecture Notes in Computer Science*, págs. 1271–1282. Springer Verlag, Berlin Heidelberg, 2003.

[297] M. Pelikan, D.E. Goldberg, y E. Cantú-Paz. BOA: The bayesian optimization algorithm. En [33], págs. 525–532.

[298] M. Pelikan, M.W. Hauschild, y F.G. Lobo. Estimation of distribution algorithms. En J. Kacprzyk y W. Pedrycz, eds., *Handbook of Computational Intelligence*, págs. 899–928. Springer Verlag, Berlin Heidelberg, 2015.

[299] M. Pelikan, K. Sastry, y E. Cantú-Paz, eds. *Scalable Optimization via Probabilistic Modeling*, tomo 33 de *Studies in Computational Intelligence*. Springer Verlag, Berlin Heidelberg, 2006.

[300] C.B. Pettey, M.R. Leuze, y J.J. Grefenstette. A parallel genetic algorithm. En *Second International Conference on Genetic Algorithms on Genetic Algorithms and Their Application*, págs. 155–161. L. Erlbaum Associates Inc, NJ, USA, 1987.

[301] J. Puchinger, G.R. Raidl, y G. Koller. Solving a real-world glass cutting problem. En J. Gottlieb y G.R. Raidl, eds., *4th European Conference on Evolutionary Computation in Combinatorial Optimization*, tomo 3004 de *Lecture Notes in Computer Science*, págs. 165–176. Springer-Verlag, Berlin Heidelberg, 2004.

[302] D. Quade. Using weighted rankings in the analysis of complete blocks with additive block effects. *Journal of the American Statistical Association*, 74:680–683, 1979.

[303] E. Radetic, M. Pelikan, y D.E. Goldberg. Effects of a deterministic hill climber on hBOA. En *Genetic and Evolutionary Computation*, págs. 437–444. ACM Press, New York, NY, USA, 2009.

[304] J. Ramanujam y P. Sadayappan. Tiling multidimensional iteration spaces for non shared memory machines. *Supercomputing*, 91:111–120, 1991.

[305] C. Ramsey y J.J. Grefensttete. Case-based initialization of genetic algorithms. En S. Forrest, ed., *Fifth International Conference on Genetic Algorithms*, págs. 84–91. Mor, San Mateo, CA, 1993.

[306] I. Rechenberg. *Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution*. Frommann-Holzboog Verlag, Stuttgart, 1973.

[307] C.R. Reeves, ed. *Modern Heuristics Tecnniques for Combinatorial Problems*. John Wiley & Sons, 1993.

[308] C.R. Reeves. Using genetic algorithms with small populations. En S. Forrest, ed., *Fifth International Conference on Genetic Algorithms*, págs. 92–99. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[309] C.R. Reeves. Hybrid genetic algorithms for bin-packing and related problems. *Annals of Operations Research*, 63(3):371–396, 1996.

[310] C.R. Reeves y J.E. Rowe. *Genetic Algorithms: Principles and Perspectives*. Kluwer, Norwell, MA, 2001.

[311] T. Reiko. Distributed genetic algorithms. En *3rd International Conference on Genetic Algorithms*, págs. 434–439. Morgan Kaufmann Publishers, CA, USA, 1989.

[312] A. Réka y A.L. Barabási. Statistical mechanics of complex networks. *Review of Modern Physics*, 74(1):47–97, 2002.

[313] V. Robles, M. Miguel, y P. Larrañaga. Solving the traveling salesman problem with EDAs. En P. Larra naga y J.A. Lozano, eds., *Estimation of*

Distribution Algorithms, A New Tool for Evolutionary Computation, tomo 2 de *Genetic Algorithms and Evolutionary Computation*, págs. 221–229. Springer Verlag, Berlin Heidelberg, 2002.

[314] G. Rudolph. On takeover times in spatially structured populations: Array and ring. En K.K. Lai et al., eds., *Genetic Algorithms and Applications*, págs. 144–151. Global-Link Publishing Company, 2000.

[315] G. Rudolph. Takeover times and probabilities of non-generational selection rules. En [364], págs. 903–910.

[316] G. Rudolph. Takeover times of noisy non-generational selection rules that undo extinction. En V. Kůrková et al., eds., *Artificial Neural Nets and Genetic Algorithms*, págs. 268–271. Springer Verlag, Berlin Heidelberg, 2001.

[317] G. Rudolph. Takeover time in parallel populations with migration. En B. Filipic y J. Silc, eds., *Second International Conference on Bioinspired Optimization Methods and their Applications - BIOMA 2006*, págs. 63–72. 2006.

[318] G. Rudolph y J. Sprave. A cellular genetic algorithm with self-adjusting acceptance threshold. En *1st IEE/IEEE International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*, págs. 365–372. London, UK, 1995.

[319] J. Sarma y K. de Jong. An analysis of local selection algorithms in a spatially structured evolutionary algorithm. En T. Bäck, ed., *7th International Conference on Genetic Algorithms*, págs. 181–186. Morgan Kaufmann Publishers, East Lansing, MI, USA, 1997.

[320] L.F.G. Sarmenta. Bayanihan: Web-based volunteer computing using java. En Y. Masunaga, T. Katayama, y M. Tsukamoto, eds., *Worldwide Computing and Its Applications - WWCA 1998*, tomo 1368 de *Lecture Notes in Computer Science*, págs. 444–461. Springer-Verlag, Berlin Heidelberg, 1998.

[321] Y. Sato y M. Sato. Parallelization and fault-tolerance of evolutionary computation on many-core processors. En *IEEE Congress on Evolutionary Computation*, págs. 2062–2609. IEEE Press, 2013.

[322] B. Schönfisch y A. de Roos. Synchronous and asynchronous updating in cellular automata. *Biosystems*, 51(3):123–143, 1999.

[323] C. Schumacher, M.D. Vose, y L.D. Whitley. The no free lunch and description length. En [336], págs. 565–570.

[324] H.P. Schwefel. *Numerische Optimierung von Computer-Modellen Mittels der Evolutionsstrategie*, tomo 26 de *Interdisciplinary Systems Research*. Birkhäuser Verlag, 1977.

[325] H.P. Schwefel. *Numerical Optimisation of Computer Models*. Wiley, New York, 1981.

[326] H.P. Schwefel. Imitating evolution: Collective, two-level learning processes. En *Explaining Process and Change - Approaches to Evolutionary Economics*, págs. 49–63. University of Michigan Press, Ann Arbor, Michigan, 1992.

[327] A.M. Sintes. Recent results on the search for continuous sources with ligo and geo 600. *Journal of Physics: Conference Series*, 39:36–38, 2005.

[328] M. Sipper y C.A. Peña Reyes. Evolutionary computation in medicine: An overview. *Artificial Intelligence in Medicine*, 19(1):1–23, 2000.

[329] Z. Skolicki y K.D. Jong. The influence of migration sizes and intervals on island models. En H.G. Beyer, ed., *Genetic and Evolutionary Computation - GECCO 2005*, págs. 1295–1302. ACM Press, New York, NY, USA, 2005.

[330] J. E. Smith. Co-evolution of memetic algorithms: Initial investigations. En [243], págs. 537–548.

[331] J. E. Smith. Coevolving memetic algorithms: A review and progress report. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 37(1):6–17, 2007.

[332] J.E. Smith. The Co-Evolution of memetic algorithms for protein structure prediction. En [170], págs. 105–128.

[333] J.E. Smith. Self-adaptation in evolutionary algorithms for combinatorial optimisation. En [76], págs. 31–57.

[334] J.E. Smith. Self-adaptative and coevolving memetic algorithms. En [263], págs. 167–188.

[335] C. Snijders, U. Matzat, y U.D. Reips. "Big Data": Big gaps of knowledge in the field of internet science. *International Journal of Internet Science*, 7(1):1–5, 2012.

[336] L. Spector et al., eds. *Genetic and Evolutionary Computation - GECCO 2000*. Morgan Kaufmann Publishers, San Francisco, CA, USA, 2001.

[337] J. Sprave. A unified model of non-panmictic population structures in evolutionary algorithms. En *IEEE Congress on Evolutionary Computation*, págs. 1384–1391. IEEE Press, 1999.

[338] R. Sterritt y D. Bustard. Towards an autonomic computing environment. En *14th International Workshop on Database and Expert Systems Applications*, págs. 694–698. 2003.

[339] R. Storn y K. Price. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, 1997. Kluwer Academic Publishers.

[340] D. Stutzbach y R.Rejaie. Understanding churn in peer-to-peer networks. En *6th ACM SIGCOMM Conference on Internet Measurement - IMC 2006*, págs. 189–202. ACM Press, New York, NY, USA, 2006.

[341] D. Sudholt. Memetic algorithms with variable-depth search to overcome local optima. En C. Ryan y M. Keijzer, eds., *Genetic and Evolutionary Computation Conference - GECCO 2008*, págs. 787–794. ACM Press, Atlanta, GA, 2008.

[342] D. Sudholt. The impact of parametrization in memetic evolutionary algorithms. *Theoretical Computer Science*, 410:2511–2528, 2009.

[343] D. Sudholt. Parametrization and balancing local and global search. En [263], págs. 55–72.

[344] D. Sudholt. Parallel evolutionary algorithms. En J. Kacprzyk y W. Pedrycz, eds., *Handbook of Computational Intelligence*, págs. 929–959. Springer-Verlag, Berlin Heidelberg, 2014.

[345] Y. Tenne. Memetic algorithms in the presence of uncertainties. En [263], págs. 223–241.

[346] S. Thakkar, M. Dubois, A.T. Laundrie, G.S. Sohi, D.V. James, S. Gjessing, M.Tapar, B. Delagi, M. Carton, y A. Despain. New directions in scalable shared-memory multiprocessor architectures. *IEEE Computer*, 23(6):71–83, 1990.

[347] D. Thierens et al., eds. *Genetic and Evolutionary Computation - GECCO 2007*. ACM Press, New York, NY, USA, 2007.

[348] M. Tomassini. *Spatially Structured Evolutionary Algorithms*. Natural Computing Series. Springer Verlag, Berlin Heidelberg, 2005.

[349] A.M. Turing. On computable numbers, with an application to the entscheidungs problem. *London Mathematical Society*, 2(42):230–265, 1936.

[350] A.M. Turing. On computable numbers, with an application to the entscheidungs problem: A correction. *London Mathematical Society*, 43(6):544–546, 1938.

[351] A.M. Turing. Intelligent machinery. *Reprinted in Cybernetics: Key Papers (1968)*, 1948. Ed. C.R. Evans and A.D.J. Robertson.

[352] S. Varrette, E. Tantar, y P. Bouvry. On the resilience of [distributed] EAs against cheaters in global computing platforms. En *25th IEEE International Symposium on Parallel and Distributed Processing - IPDPS 2011*, págs. 409–417. IEEE Press, Anchorage, Alaska, 2011.

[353] John von Neumann. *First Draft of a Report on the EDVAC*, 1945. URL https://web.archive.org/web/20130314123032/http://qss.stanford.edu/~godfrey/vonNeumann/vnedvac.pdf. Accesed 15/05/2015.

[354] C. Voudouris. *Guided Local Search for Combinatorial Optimization Problems*. Tesis Doctoral, Department of Computer Science, University of Essex, 1997.

[355] C. Voudouris y E. Tsang. Guided local search. *European Journal of Operational Research*, 113(2):469–499, 1999.

[356] R. Watson y J. Pollack. Hierarchically consistent test problems for genetic algorithms. En *IEEE Congress on Evolutionary Computation - CEC 99*, tomo 2, págs. 1406–1413. IEEE Press, 1999.

[357] R.A. Watson, G.S. Hornby, y J.B. Pollack. Modeling building-block interdependency. En [108], págs. 97–106.

[358] D.J. Watts y S.H. Strogatz. Collective dynamics of "small-world"networks. *Nature*, 393:440–442, 1998.

[359] M. Weber, F. Neri, y V. Tirronen. A study on scale factor in distributed differential evolution. *Information Sciences*, 181(12):2488–2511, 2011.

[360] W. Weibull. A statistical distribution function of wide applicability. *Journal of Applied Mechanics*, 18(3):293–297, 1951.

[361] J.M. Whitacre, R.A. Sarker, y Q. Pham. The self-organization of interaction networks for nature-inspired optimization. *IEEE Transactions on Evolutionary Computation*, 12:220–230, 2008.

[362] D. Whitley. A free lunch proof for gray versus binary encodings. En [33], págs. 726–733.

[363] D. Whitley y J. Watson. Complexity theory and the no free lunch theorem. *Search Methodologies*, págs. 317–339, 2005. E. K. Burke and G. Kendall, Ed. Springer-Verlag.

[364] D. Whitley et al., eds. *Genetic and Evolutionary Computation - GECCO 2000*. Morgan Kaufmann Publishers, San Francisco, CA, USA, 2000.

[365] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics*, 1(6):80–83, 1945.

[366] B. Wilkinson y M. Allen. *Parallel Programming Techniques and Applications Using Networked Workstations and Parallel Computers*. Prentice Hall, 1999.

[367] D.H. Wolpert y W.G. Macready. No free lunch theorems for search. Inf. Téc. SFI-TR-95-02-010, Santa Fe Institute, Santa Fe, NM, USA, 1995.

[368] D.H. Wolpert y W.G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(67):67–82, 1997.

[369] E.L. Yu y P.N. Suganthan. Ensemble of niching algorithms. *Information Sciences*, 180(15):2815–2833, 2010.

[370] F. Zambonelli. Exploiting biased load information in direct-neighbour load balancing policies. *Parallel Computing*, 1999.

[371] Q. Zhang, J. Sun, E. Tsang, y J. Ford. Estimation of distribution algorithm with 2-opt local search for the quadratic assignment problem. En J.A. Lozano et al., eds., *Towards a New Evolutionary Computation*, tomo 192 de *Studies in Fuzziness and Soft Computing*, págs. 281–292. Springer Verlag, Berlin Heidelberg, 2006.

[372] H. Zhao, X. Liu, y X. Li. A taxonomy of peer-to-peer desktop grid paradigms. *Journal of Cluster Computing*, 14(2):129–144, 2010.

[373] W. Zhao y H. Schulzrinne. Dotslash: A self-configuring and scalable rescue system for handling web hotspots effectively. En *International Workshop on Web Caching and Content Distribution - WCW 2004*, págs. 1–18. 2004.

[374] Z. Zhou, Y.S. Ong, P.B. Nair, A.J. Keane, y K.Y. Lum. Combining global and local surrogate models to accelerate evolutionary optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 37(1):66–76, 2007.

Índice alfabético