UNIVERSIDADE DA CORUÑA

Facultad de Informática

Departamento de Computación

Tesis Doctoral

CARACTERIZACIÓN DE ESPACIOS DE CALIDAD Y ALGORITMOS EVOLUTIVOS EN PROBLEMAS DE OPTIMIZACIÓN CON CODIFICACIÓN REAL

Pilar Caamaño Sobrino

Directores:
Francisco J. Bellas Bouza
Jose Antonio Becerra Permuy

Fecha:
Noviembre, 2010

UNIVERSIDADE DA CORUÑA

D. Francisco J. Bellas Bouza, Profesor Contratado Doctor del Departamento de Computación de la Universidade da Coruña,

D. Jose Antonio Becerra Permuy, Profesor Contratado Doctor del Departamento de Computación de la Universidade da Coruña,

CERTIFICAN:

Que la memoria titulada ``Caracterización de espacios de calidad y algoritmos evolutivos en problemas de optimización con codificación real" ha sido realizada por Dña. Pilar Caamaño Sobrino bajo nuestra dirección en el Departamento de Computación de la Universidade da Coruña, y constituye la Tesis que presenta para optar al grado de Doctor.

Fdo. Francisco J. Bellas Bouza
Codirector de la Tesis Doctoral

Fdo. Jose Antonio Becerra Permuy
Codirector de la Tesis Doctoral

Publicaciones

Durante la realización de esta tesis se han publicado los siguientes trabajos en congresos y revistas científicas:

- Duro, R.J.; Bellas, F.; Caamaño, P. and Varela, G.; Automatic model decomposition and reuse in an evolutionary cognitive mechanism; *Evolving Systems, 1* pp.:129-141, Springer, 2010.

- Prieto, A.; Bellas, F.; Caamaño, P., Duro, R. J.; Automatic Behavior Pattern Classification for Social Robots; *Hybrid Artificial Intelligence Systems (1)*, Romay, M. G.; Corchado, E. & García-Sebastián, M. T. (Eds.); Springer, 2010, 6076, 88-95.

- Caamaño, P.; Tedín, R.; Paz-López, A., Becerra, J. A.; JEAF: A Java Evolutionary Algorithm Framework; *IEEE Congress on Evolutionary Computation*, 2010, 1-8.

- Prieto, A.; Caamaño, P.; Bellas, F., Duro, R. J.; Population dynamics analysis in an agent-based artificial life system for engineering optimization problems; *IEEE Congress on Evolutionary Computation*, 2009, 2724-2731.

- García, P.; Caamaño, P.; Bellas, F., Duro, R. J.; A Behavior Based Architecture with Auction-Based Task Assignment for Multi-robot Industrial Applications; *Proceedings of the 3rd International Work-Conference on The Interplay Between Natural and Artificial Computation: Part II: Bioinspired Applications in Artificial and Natural Computation*, Mira, J. M.; Ferrández, J. M.; Álvarez, J. R.; de la Paz, F. & Toledo, F. J. (Eds.), Springer, 2009, 5602, 372-381.

- Míguez, M.; Caamaño, P.; Tedín, R.; Díaz, V. y Martínez, A.; Un sistema embarcado de evaluación de la estabilidad y ayuda al patrón de buques de pesca; *Actas del 48º Congreso de Ingeniería Marítima*, 2009.

- Míguez, M.; Caamaño, P.; Tedín, R.; Díaz, V. y Martínez, A.; Un sistema embarcado de evaluación de la estabilidad y ayuda al patrón de buques de pesca; *Ingeniería Naval*, pp.: 92-98, 2009.

- Caamaño, P.; Bellas, F.; Becerra, J. A., Duro, R. J.; Application domain study of evolutionary algorithms in optimization problems; *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, GECCO, ACM, Ryan, C. & Keijzer, M. (Eds.), 2008, 495-502.

- Prieto, A.; Bellas, F.; Caamaño, P., Duro, R. J.; A Complex Systems Based Tool for Collective Robot Behavior Emergence and Analysis; *Hybrid Artificial Intelligence Systems*, Corchado, E.; Abraham, A. & Pedrycz, W. (Eds.), Springer, 2008, 5271, 633-640.

- Caamaño, P.; Becerra, J. A.; Bellas, F., Duro, R. J.; Using Spiking Neural Networks for the Generation of Coordinated Action Sequences in Robots; *Proceedings of the 15th international conference on Advances in neuro-information processing (1)*, Köppen, M.; Kasabov, N. K. & Coghill, G. G. (Eds.), Springer, 2008, 5506, 1013-1020.

- Míguez Gonzalez, M.; Caamaño Sobrino, P.; Díaz Casás, V. y Martínez López, A.; Implicaciones de la resolución IMO MSC.194(80) en el Diseño de Buques Ro-Pax; *Actas del 47º Congreso de Ingeniería Marítima*, 2008.

- Caamaño, P.; Prieto, A.; Becerra, J. A.; Duro, R. J., Bellas, F.; Evolutionary Tool for the Incremental Design of Controllers for Collective Behaviors; *Proceedings of the 2nd international work-conference on The Interplay Between Natural and Artificial Computation, Part I: Bio-inspired Modeling of Cognitive Tasks*, Mira, J. & Álvarez, J. R. (Eds.), Springer, 2007, 4527, 587-596.

- Caamaño, P.; Becerra, J. A.; Duro, R. J., Bellas, F.; Incremental Evolution of Stigmergy-Based Multi Robot Controllers Through Utility Functions; *Knowledge-Based Intelligent Information and Engineering Systems and the XVII Italian Workshop on Neural Networks on Proceedings of the 11th International Conference (2*), Apolloni, B.; Howlett, R. J. & Jain, L. C. (Eds.), Springer, 2007, 4693, 1187-1195.

- Caamaño, P.; Prieto, A.; Bellas, F.; Becerra, J.A. y Duro, R.J.; BDesigner: Una Herramienta Evolutiva para la Creación de Comportamientos Colaborativos; *Actas del Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB)*, 2007.

Agradecimientos

Cuando termine de escribir estos agradecimientos me daré cuenta de que faltarán muchos nombres. Todos los que aparecen tienen un hueco merecido, pero también faltarán otros porque el espacio es reducido. Es difícil resumir estos cuatro años. Sin el apoyo, tanto profesional como personal, de cada uno de los que aquí aparecen, esta tesis no hubiese llegado a buen puerto. Retocaré estas líneas un millón de veces antes de la versión definitiva, pero el significado final será el mismo. Sea como sea que esté escrito, simplemente, gracias.

Hay cuatro personas sin las cuales esta tesis no hubiese existido nunca. Mis directores de tesis, Fran y Jose Antonio. A ellos tengo que agradecerles el haberme permitido el lujo de seguir aprendiendo sobre aquello que me gusta mientras trabajo, a día de hoy es algo que no tiene precio. Su paciencia, en algunas ocasiones infinita, y el apoyo, tanto profesional como personal durante estos cuatro años. Gracias por todo lo que he aprendido en este proceso y todo lo que queda por aprender. Mis padres, Pili y José Ramón, porque ellos son los responsables de mi educación, porque nunca me cortaron las alas en el ansia de estudiar, aprender y trabajar en lo que me gusta y aquello que me hace feliz. Por animarme siempre a seguir adelante y aguantar, con infinita paciencia, mis continuos cambios de humor durante este tiempo.

Además de a ellos, quiero agradecer el apoyo y la confianza de Richard Duro que, como coordinador del GII, es también responsable y parte importante de este trabajo. Gracias por "invitarme" a pertenecer a este grupo.

El ambiente de trabajo envidiable en el cual se ha desarrollado esta tesis es responsabilidad de mis compañeros de laboratorio. Gracias a todos ellos, a los que han estado desde el principio, a los que pasaron por aquí y, también, a los que han ido llegando a lo largo de estos años.

El desarrollo de una tesis, como la vida, es una montaña rusa de subidas y bajadas. Que vuelva a subir cada vez que bajo se lo debo a los que tengo alrededor. A mis hermanos y a mi tío Fernando, porque, a pesar de la distancia y las diferencias, siempre estáis. A Lola, María, Elena, Simón, etc., gracias. Sois, con mi familia, mi punto de apoyo constante, ese lugar al que siempre quiero volver *(43º 29' 18", -8º 19' 17")*.

Y una lista infinita de nombres: toda mi familia, del primero al último, la parte de mi madre y la parte de mi padre, en especial a mi abuela. Cali, Papisa, Alejandra, Luis, Jorge, Inés, Marta (Malde), Paula, Verto (con uve), Fran, Rocío, Carmen, Gundar, Marco, todos mis compañeros de facultad, Mónica, Fran, María, Juan Pedro, Elkin, Daniel, Pedro, Tiago o, lo que es lo mismo, Notthingham.

Ferrol, 29 de Noviembre del 2010.

Índice general

Capítulo 1

Resumen

Esta tesis doctoral propone un procedimiento de caracterización formal de algoritmos evolutivos y espacios de calidad en problemas de optimización con codificación real. La principal motivación para el desarrollo de este tema ha sido la constatación de que el gran auge experimentado en la aplicación de los algoritmos evolutivos en problemas reales cada vez más complejos ha implicado el desarrollo de nuevas técnicas más avanzadas y con mejores resultados pero, sin embargo, este gran nivel de actividad no ha venido acompañado de un análisis formal de dichas técnicas. En consecuencia, actualmente los usuarios de algoritmos evolutivos no expertos en el campo poseen gran cantidad de opciones y variantes de las mismas, pero carecen de información objetiva sobre el ámbito de aplicación de cada una de ellas. En esta tesis doctoral se ha llevado a cabo una primera aproximación al desarrollo de un procedimiento de caracterización formal destinado a los diseñadores de algoritmos evolutivos para que, a la hora de presentar sus trabajos a la comunidad científica, utilicen una metodología común que posibilite la caracterización práctica del algoritmo desde un punto de vista totalmente objetivo y con conclusiones que sean fácilmente utilizables por parte de los usuarios.

El procedimiento de caracterización que se describe a lo largo de la tesis va, por tanto, dirigido a usuarios expertos en computación evolutiva, es decir, desarrolladores de algoritmos. Pero los pasos concretos a seguir se presentan desde una perspectiva muy básica, partiendo de una actitud crítica a los principios establecidos hasta el momento y tratando de asumir métodos claramente validados. Finalmente, el procedimiento de caracterización se compone de tres etapas claramente diferenciadas e igualmente relevantes para obtener un análisis que aporte información objetiva. Primeramente, se debe obtener un conjunto de funciones de prueba representativo y este debe ser convenientemente caracterizado. Además, se deben establecer unas medidas de error y rendimiento que permitan extraer conclusiones válidas y comparar algoritmos. Por último, es necesario formalizar las condiciones de ejecución del algoritmo evolutivo y proponer una metodología de análisis formal de los resultados obtenidos. A lo largo de esta memoria se explica cómo se ha llegado a establecer estas tres etapas y se propone una posible implementación de cada una de ellas.

El campo de la Computación Evolutiva y, más concretamente, el campo de los Algoritmos Evolutivos es relativamente joven si lo comparamos con otras áreas de investigación dedicadas a la optimización de funciones. Su origen se remonta a la década de los 60 y principios de los 70, donde aparecieron como técnicas avanzadas de búsqueda para espacios n-dimensionales que hacían uso de cálculos intensivos favorecidos por el avance en las capacidades de cómputo. Su utilización se popularizó durante la década de los 80 debido a su gran

éxito a la hora de resolver tanto problemas matemáticos abstractos (problema del viajante, problema de la mochila, etc.) como problemas más concretos como, por ejemplo, control de flujo, análisis de patrones, etc. En estas tres décadas se establecieron cuatro paradigmas básicos: los Algoritmos Genéticos [Holland, 1975], las Estrategias Evolutivas [Rechenberg, 1973], la Programación Evolutiva [Fogel et al., 1966] y la Programación Genética [Koza, 1992b]. A lo largo de la década de los 90, su utilización fue en aumento con gran éxito en la optimización de funciones complejas y, sobre todo, en las distintas ramas de la Inteligencia Artificial, que utilizaban los algoritmos evolutivos como técnicas de aprendizaje automático. En la última década, estos algoritmos han experimentado un auge todavía mayor debido al aumento en las capacidades de cómputo, logrando así reducir los tiempos de cálculo que implican y abriendo un amplio abanico de campos de aplicación en los que dichos tiempos son cruciales, fundamentalmente, en las distintas ramas de la ingeniería. Son numerosas las aplicaciones reales en las que hoy en día se utilizan algoritmos evolutivos para el diseño automático, para la optimización en problemas de muy alta dimensionalidad, para la optimización en funciones multiobjetivo, para sistemas de control con gran cantidad de parámetros, etc.

Debido a esto, a nivel científico la productividad ha sido también muy elevada. Así, es común encontrar en las revistas especializadas trabajos dedicados a la presentación de resultados de aplicación de algoritmos evolutivos en diferentes campos, así como el desarrollo de nuevas técnicas para problemas concretos. Las competiciones de comparación de algoritmos evolutivos son también cada vez más numerosas en los congresos y conferencias del área (por ejemplo, en el congreso internacional de la IEEE *Congress of Evolutionary Computation* (CEC) y en la *Genetic and Evolutionary Computation Conference* (GECCO)).

Una de las consecuencias de esta gran actividad ha sido que usuarios no expertos en computación evolutiva han comenzado a ver en estas técnicas una opción realista y simple de utilizar. De hecho, una de las principales ventajas de este tipo de algoritmos y, a la vez, una de las principales claves de su éxito, es que trabajan como una "caja negra", es decir, el algoritmo no requiere ningún conocimiento explícito del problema a resolver. Son métodos de optimización de orden cero cuyo único requisito es la posibilidad de evaluar la función objetivo que define el problema. Es en este punto donde hemos detectado un vacío y un problema a nivel formal que se debe afrontar, y es que estos usuarios no expertos tienen que seleccionar el algoritmo evolutivo prácticamente ``a ciegas". Es decir, de entre la multitud de opciones existentes, se suele realizar un proceso de prueba y error con distintos algoritmos evolutivos hasta que uno resuelve el problema.

Como se mostrará a lo largo del capítulo de revisión de trabajos previos, los desarrolladores de algoritmos no comparan objetivamente sus técnicas, sino que realizan pruebas tratando de enfatizar los puntos fuertes de las mismas. Por tanto, existe información sobre los dominios de aplicación concretos en los que un cierto algoritmo ha resultado exitoso, pero no sobre los que presentan problemas. En este sentido, en las competiciones comentadas anteriormente, el resultado suele ser un algoritmo ``campeón" en un cierto tipo de funciones objetivo, pero sin proporcionar más detalles acerca del mismo. Si bien no deja de crecer el interés por obtener el ``mejor" algoritmo en estas competiciones y usar dicho algoritmo, el desarrollo de una metodología adecuada para el análisis y comparación de los mismos sigue siendo una asignatura pendiente. Además, en los últimos años han surgido trabajos en los cuales se defiende la idea de que, si inicialmente se dispone de dicho conocimiento, por qué no utilizarlo en beneficio de la evolución del sistema. Esto provoca que se implementen cada vez más algoritmos "ad-hoc" para problemas muy específicos, incrementando la falta de estandarización y alejando a usuarios no expertos del campo de la evolución. En este sentido, el procedimiento de caracterización que se describe en esta tesis se presenta como apoyo a aquellos trabajos

que defienden que no es necesario incluir conocimiento del dominio, sino que una caracterización formal y adecuada de los algoritmos permitiría la elección y configuración de uno u otro dependiendo del tipo del problema a resolver sin necesidad de implementar algoritmos específicos.

Una vez detallada la motivación inicial y la aportación científica que se pretende con esta tesis doctoral, a continuación se realizará un breve resumen del contenido de esta memoria:

El objetivo principal de esta tesis y los sub-objetivos necesarios para alcanzarlo se detallan en el capítulo 2.

A continuación se enmarcará este trabajo dentro del estado actual del campo de la computación evolutiva. Para ello, en una primera sección del capítulo 3, se explicarán brevemente los conceptos teóricos básicos que se manejarán a lo largo de la memoria, introduciendo el paradigma de los algoritmos evolutivos, de dónde surgen y el por qué de su éxito, cuáles son los mecanismos que gobiernan su comportamiento y en qué campos han sido utilizados. En la segunda sección de dicho capítulo, se realizará una revisión de otros trabajos relacionados con el análisis y la comparación formal de algoritmos evolutivos para, de esta forma, incidir en la falta de estandarización del campo y en la necesidad de realizar avances en esta línea.

El procedimiento de caracterización que constituye el tema central de la tesis se explicará en detalle en el capítulo 4. En él se justificará la necesidad y se desarrollarán en profundidad las tres etapas que conforman el procedimiento. La primera de las etapas está centrada en la selección y caracterización de un conjunto de funciones de prueba, y se describe en la primera sección del capítulo. Se comienza por el estudio de las superficies de calidad y su relevancia en el funcionamiento de los algoritmos evolutivos, explicando la problemática que supone no tener en cuenta las características detalladas de estas superficies. Se lleva a cabo, a continuación, una revisión bibliográfica de los trabajos relacionados con el concepto de superficie de calidad para determinar cuáles son las características topológicas que, desde el punto de vista de los algoritmos evolutivos, condicionan en mayor medida su funcionamiento. Tras este proceso se concluye, en una primera aproximación, que estas son la separabilidad y la modalidad. Finalmente, en esta sección se detalla el conjunto de funciones prueba que serán utilizadas en la sección de aplicación del procedimiento de caracterización y que también se propone como base para que otros investigadores analicen sus algoritmos. En la última parte de la primera sección de este capítulo 4, se presentarán los algoritmos de caracterización de superficies de calidad en términos de separabilidad y modalidad desarrollados. Estos algoritmos serán aplicados a las funciones de prueba establecidas anteriormente con el fin de estimar sus características topológicas en base a las cuales se caracterizarán los algoritmos evolutivos.

La segunda etapa del procedimiento de caracterización desarrollado se centra en estudiar la problemática de la selección de las medidas de error y rendimiento utilizadas en el análisis el comportamiento de los algoritmos, y se encuentra en la segunda sección del capítulo 4. A lo largo de la misma se realizará una breve revisión de las medidas más comunes, explicando sus ventajas y desventajas y se presentará la medida *CPEM* (Combined Performance and Error Measure), desarrollada en el marco de esta tesis y utilizada como base en el análisis de los resultados.

Las dos primeras etapas del procedimiento de caracterización se pueden realizar de forma concurrente, ya que abordan dos problemas independientes, por un lado el establecimiento de un "benchmark" relevante y formalmente caracterizado, y por otro la selección de medidas de error y rendimiento adecuadas. Estas dos etapas no tienen por qué ser llevadas a cabo por el desarrollador de algoritmos evolutivos de cara a caracterizar su implementación, pu-

diendo hacer uso de conjuntos de funciones o medidas de error ya existentes, como las que se proporcionan en esta tesis doctoral, u otras. La tercera etapa del procedimiento de caracterización, el estudio experimental, sí es responsabilidad del desarrollador en todos los casos y consiste en la ejecución del algoritmo sobre el conjunto de funciones de prueba seleccionado y en el análisis formal de los resultados. Para ello, en esta etapa se deben establecer una serie de condiciones de ejecución que permitan obtener resultados objetivamente comparables entre algoritmos y se debe plantear una metodología de análisis de resultados en base a las características establecidas en la primera etapa sobre el conjunto de funciones de prueba.

En la tercera sección del capítulo se proponen soluciones a ambos aspectos. En cuanto a las condiciones de ejecución, se establece que todos los parámetros propios del algoritmo evolutivo los fija el desarrollador para un rendimiento óptimo, a excepción del tamaño de la población que, dada su gran influencia en el comportamiento de los algoritmos evolutivos, debe ser seleccionado tras un barrido controlado. También se establecen valores concretos sobre el criterio de parada, número de ejecuciones a realizar, medidas a realizar, etc. En cuanto al análisis de los resultados obtenidos de la ejecución sobre el conjunto de funciones de prueba, en esta sección se describe la metodología propuesta, que se ha dividido en tres etapas: análisis de los resultados con la configuración básica del algoritmo, análisis modificando los parámetros iniciales y análisis comparativo. En estas tres etapas, el análisis de los resultados se lleva a cabo en función de la respuesta del algoritmo a la separabilidad y modalidad de las funciones. Una vez establecidas conclusiones en este sentido, estas se relacionan con las cuatro propiedades básicas que condicionan el comportamiento los algoritmos evolutivos: el balance exploración / explotación, la presión selectiva, las direcciones de búsqueda y el tamaño del paso de mutación. En esta tercera sección del capítulo se explican cada una de estas propiedades dejando claro cómo influyen en el rendimiento de los algoritmos y en qué medida afectan a cada uno de los tipos de funciones que componen el conjunto de prueba.

Para terminar, el capítulo 4 finaliza con el planteamiento resumido de las etapas que constituyen el procedimiento de caracterización.

En el capítulo 5 se aplicará el procedimiento de caracterización presentado en el capítulo anterior sobre cuatro algoritmos evolutivos. Estos algoritmos son: un *Algoritmo Genético* con operadores de codificación real, el algoritmo *Covariance Matrix Adaptation*, el algoritmo *Differential Evolution* y los algoritmos *Macroevolutivos*. Los tres primeros se han seleccionado por ser una referencia en el contexto actual de la computación evolutiva y el cuarto por ser un algoritmo más desconocido, y por tanto, un buen ejemplo de cómo caracterizar un nuevo desarrollo. De cada uno de ellos se explicarán sus características básicas incidiendo en detalle en su estrategia de búsqueda, ya que nos ayudará a comprender su comportamiento ante las funciones objetivo utilizadas. A continuación se aplicará el procedimiento de caracterización tal y como lo debería hacer un desarrollador. Las dos primeras etapas implican seleccionar el conjunto de funciones de prueba y establecer las medidas de error y rendimiento. En este caso, se utilizarán las explicadas en el capítulo 4 que serán comunes a los cuatro algoritmos a caracterizar. La tercera etapa se repetirá para cada algoritmo, comenzando por la realización de un análisis poblacional de cada uno de ellos con objeto de fijar el número de individuos óptimo en cada función y dimensión. Como se verá, cada algoritmo evolutivo tiene unos requisitos poblacionales diferentes que dependen de su estrategia de búsqueda y de las propiedades de las funciones a resolver. La parte central de cada sección del capítulo 5 consiste en el análisis de los resultados obtenidos para el tamaño de población óptimo, clasificándolos en base a la separabilidad y modalidad de las funciones. Una vez detectadas aquellas funciones que los algoritmos no son capaces de resolver con la configuración utilizada, se analizará el comportamiento de los algoritmos para tratar de ajustar los parámetros

y mejorar los resultados obtenidos. La última parte de este capítulo se centrará en el análisis comparativo de los resultados de cada algoritmo.

Para terminar, en el capítulo 6 se exponen las principales conclusiones del trabajo desarrollado en esta tesis doctoral, incidiendo en las aportaciones llevadas a cabo en cuanto a la caracterización formal de los algoritmos evolutivos. En el capítulo 7 se presentan las líneas de trabajo que se abren en el futuro y los principales temas que han quedado por resolver.

Cabe destacar que la realización de esta tesis se enmarca en una serie de proyectos del Grupo Integrado de Ingeniería de la Universidade da Coruña relacionados con la utilización de algoritmos evolutivos en problemas de optimización en ingeniería y financiados por el MEC, el MICINN y la Xunta de Galicia, cuya contribución ha resultado fundamental para la realización de esta tesis. El trabajo se ha llevado a cabo en los laboratorios del Grupo Integrado de Ingeniería en Ferrol y en el laboratorio ASAP de la Universidad de Nottingham (UK).

Capítulo 2

Objetivos

El objetivo principal de esta tesis doctoral es el siguiente:

El desarrollo de un procedimiento de caracterización que permita analizar formalmente el comportamiento de los algoritmos evolutivos en problemas de optimización con codificación real. Se pretende establecer una metodología de caracterización que pueda ser utilizada por los desarrolladores de estos algoritmos de tal forma que los usuarios no expertos en computación evolutiva tengan criterios objetivos para seleccionar el algoritmo más adecuado en función de las características de un problema concreto.

De cara a cumplir este objetivo global, se deberán alcanzar los siguientes sub-objetivos:

1. Determinar los problemas que deben ser abordados de cara a realizar una caracterización formal de un algoritmo evolutivo que aporte información práctica a los usuarios finales.
2. Organizar estos problemas en etapas claramente diferenciadas que puedan ser realizadas de forma práctica.
3. Estudiar con detalle cada etapa en el marco actual del campo de la Computación Evolutiva.
4. Proponer soluciones concretas a los problemas planteados en cada una de las etapas, bien utilizando propuestas ya existentes y validadas, o bien proponiendo nuevos métodos.
5. Demostrar que el procedimiento de caracterización, con las soluciones concretas que se proponen, aporta resultados formales relevantes e información útil para los usuarios de algoritmos evolutivos.

Como consecuencia del cumplimiento de estos objetivos y subobjetivos, se pretende resaltar la gran importancia de llevar a cabo una caracterización formal de los algoritmos evolutivos como requisito indispensable para que los desarrollos en este campo sean utilizables en otras ramas de la Ciencia.

El conjunto concreto de algoritmos, técnicas y métodos que se proponen en esta tesis no constituyen el objetivo central de la misma, siendo una primera aproximación a la metodología de desarrollo y análisis que se debería seguir, y un requisito necesario para poder validar el procedimiento propuesto.

Capítulo 3

Marco teórico

En este capítulo se enmarca el presente trabajo dentro del campo de la computación evolutiva con el objetivo fundamental de mostrar qué aporta a nivel científico en este ámbito y de justificar la idoneidad de las aproximaciones en las que se inspira. El capítulo se ha dividido en tres secciones principales: una primera centrada en los fundamentos teóricos de los algoritmos evolutivos y su expansión a nivel de aplicación en los últimos años, una segunda donde se revisarán los principales trabajos dedicados al análisis y comparación de algoritmos evolutivos en problemas de codificación real y su escasa repercusión práctica y, finalmente, una tercera centrada en las conclusiones del estudio del campo que servirá como punto de arranque para el desarrollo del procedimiento de caracterización.

3.1 Algoritmos evolutivos

El objetivo de esta sección es realizar una revisión actualizada del campo de la computación evolutiva y, más concretamente, de los algoritmos evolutivos en su aplicación práctica durante las dos últimas décadas. Para ello, en primer lugar se realizará una breve introducción a los elementos básicos de estos algoritmos, incidiendo en la influencia práctica que tienen sobre el funcionamiento de los mismos. A continuación, se realizará un breve resumen de los trabajos originales que llevaron al desarrollo de esta técnica de optimización y de como surgieron cuatro paradigmas con un comportamiento general común pero con características diferenciadoras importantes. La tercera subsección es clave dentro del enfoque de este trabajo, y estará dedicada a mostrar el enorme incremento en el número y tipología de problemas reales a los que los algoritmos evolutivos han sido aplicados en los últimos años. Para terminar, se revisarán los desarrollos más actuales y las tendencias futuras de estos algoritmos.

3.1.1 Conceptos básicos

El término algoritmo evolutivo (AE) engloba una serie de técnicas o paradigmas que se basan en la evolución natural para resolver problemas de optimización y búsqueda. Para implementar un AE son necesarios una serie de componentes básicos. En primer lugar, una **población** de individuos donde cada uno de ellos representa una solución candidata al problema a resolver. Cada **individuo** está formado por uno o varios **cromosomas** que son una colección o lista de genes. Los **genes** representan incógnitas del problema y se pueden utilizar

distintos tipos de codificaciones para representarlas como, por ejemplo, codificación binaria, codificación utilizando números reales o mediante caracteres. Además, cada individuo tiene asociado un **valor de calidad** o aptitud que es utilizado para establecer lo bien o mal que se comporta, comparativamente, ante un problema dado (ver figura 3.1).

En una primera fase se genera una población de individuos de forma aleatoria, tratando de cubrir la mayor área posible del espacio de búsqueda. A partir de esta población inicial, se generan nuevas soluciones utilizando **operadores evolutivos** de selección, reproducción y reemplazo. Los operadores de **selección** son los encargados de escoger, dentro de la población de padres, los individuos que pasarán a la fase de reproducción. Los operadores de **reproducción**, que se dividen en operadores de cruce y mutación, son los encargados de buscar nuevas soluciones generando una población de descendientes a partir de la sub-población de padres seleccionada por los operadores de selección. El comportamiento típico de los operadores de **cruce** consiste en generar dos descendientes a partir de dos padres combinando el material genético de los padres. Estos operadores combinan el material genético existente en la población y, por lo tanto, no se introduce, en principio, nuevo material genético. Los operadores de cruce más típicos son los cruces por puntos: cruce por un punto, cruce por dos puntos o cruce por n puntos. Los operadores de mutación más comunes generan un descendiente a partir de un padre cambiando aleatoriamente el valor de algunos de sus genes, es decir, introducen nueva información en la población. Por último, los operadores de **reemplazo** son los encargados de seleccionar los individuos que sobrevivirán y formarán la población de padres de la siguiente generación, como se explicará más adelante. El tipo de reemplazo depende del paradigma utilizado. En algunos casos sobreviven los mejores, ya sean padres o descendientes, en lo que se conoce como modelo de reemplazo con superposición. En otros casos sólo sobreviven los mejores descendientes, constituyendo los modelos de reemplazo sin superposición. [Sarma and De Jong, 1997].

En resumen, el ciclo básico de un algoritmo evolutivo incluye las siguientes fases (ver figura 3.2):

- Una **fase inicial** en la que se genera una población, normalmente aleatoria, de N individuos, que representarán las posibles soluciones iniciales a un problema dado, también llamada *población inicial de padres*.
- Una segunda fase de **evaluación** de la población inicial de padres.
- El **ciclo o proceso evolutivo**, que es el encargado de realizar el proceso de optimización en sí. Incluye las siguientes subfases:
  - A partir de la *población de padres* se seleccionarán los candidatos para su reproducción utilizando un operador de **selección**.
  - En la fase de **reproducción** se genera la *población de descendientes* a partir de la subpoblación de padres seleccionada.
  - Se evalúa la *población de descendientes*.
  - En la fase de **reemplazo** se genera la población de padres de la siguiente generación.

El problema de la representación: directa vs. indirecta

La representación de los cromosomas de los individuos es un aspecto clave de los AEs. Para explicar este problema es necesario, en primer lugar, explicar la diferencia entre genotipo

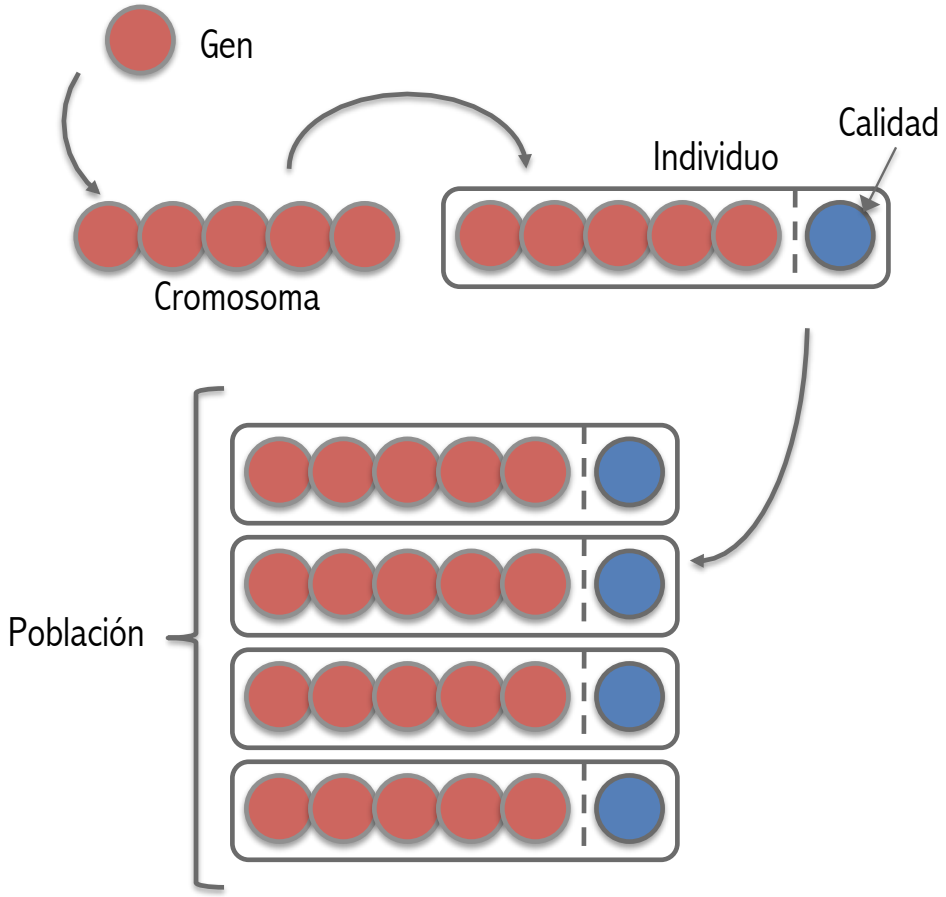


Figura 3.1: Un gen es la unidad mínima de información en un AE, la unión de varios genes forma un cromosoma. Un individuo está formado por uno o varios cromosomas y un valor de calidad. La población del AE está formado por un conjunto de individuos.

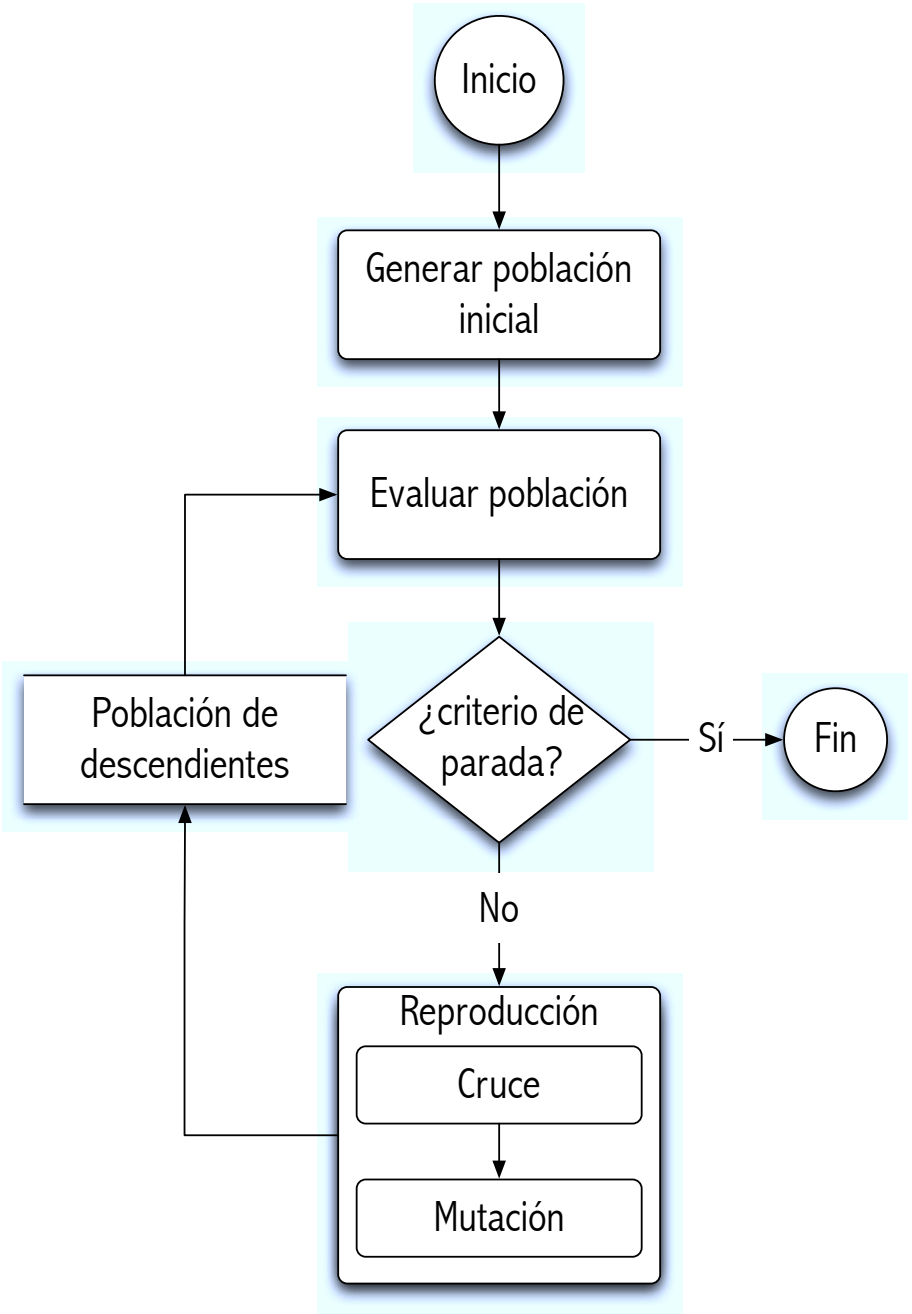


Figura 3.2: Esquema general de funcionamiento de un algoritmo evolutivo.

y fenotipo. Como ya se ha explicado anteriormente, los algoritmos evolutivos están inspirados en la evolución biológica. En este campo se define *genotipo* como la descripción genética de un individuo, es decir, el conjunto de genes y los valores de los mismos. Por otro lado, se denomina *fenotipo* a la expresión del genotipo. Dicho de otra forma, el genotipo es un conjunto de valores que contiene información sobre el individuo. Esta información "sufre" una transformación por la cual se obtiene el fenotipo en el que la información de los genes puede manifestarse o no.

A partir de estos conceptos, en Computación Evolutiva (CE) se definieron dos tipos de representaciones de cromosomas: directa e indirecta. En la representación directa, el fenotipo de un individuo es igual al genotipo, es decir, los individuos representan soluciones que internamente son exactas a su representación "externa". Por ejemplo, si se trata de un problema de optimización de parámetros, los genes de los cromosomas representan los valores de dichos parámetros en el espacio correspondiente ya sean números reales, enteros, etc. Si el problema a resolver consiste en la evolución de un árbol o un grafo, cada gen representará un nodo. Por otro lado, en la representación indirecta, el genotipo y el fenotipo son diferentes, existe una función de transformación que decodifica el genotipo. En CE es muy común utilizar la codificación binaria (cromosomas donde los genes toman valores 0 o 1) como codificación universal y a la hora de evaluar los individuos utilizar una función de transformación que decodifique los genes binarios en, por ejemplo, parámetros reales. Este no es el único ejemplo de codificación indirecta. Cuando se evolucionan redes de neuronas artificiales es posible codificar su estructura y sus parámetros en el cromosoma. En aquellos casos en los cuales se codifica la estructura, por ejemplo utilizando reglas a partir de las cuales se genera la red, se necesita una función de transformación que interprete esa información y genere la red a partir de dicho cromosoma.

La representación directa no es siempre mejor que la indirecta, ni viceversa. Existen numerosos trabajos en la bibliografía donde se realizan comparaciones de ambos tipos de representación destacando que la elección de una u otra es altamente dependiente del problema a resolver [Bentley and Kumar, 1999, Harding and Miller, 2006]. Esta decisión tiene efectos en el proceso de búsqueda que seguirá la población sobre el espacio de soluciones y en la implementación del algoritmo, concretamente en la implementación de los operadores. Existen una serie de aspectos que debemos tener en cuenta a la hora de elegir entre una u otra representación:

- La codificación directa tiene la ventaja de realizar una mayor explotación de la propiedades del problema. Los operadores pueden aprovechar las características propias del mismo para generar las nuevas soluciones. Un ejemplo es el concepto de continuidad en el caso de espacios de codificación real.

- En el caso de la codificación directa, el genotipo crece con el fenotipo pudiendo llegar a tener cromosomas de longitud muy alta que implican grandes tamaños de población y, por lo tanto, mayor tiempo de cómputo.

- Cuando existen y se conocen interacciones entre variables es más recomendable la codificación indirecta. Es decir, si, por ejemplo, tres genes poseen una relación conocida y esta puede ser formulada, es preferible codificar estos genes como uno solo y no como tres. De otra forma, el algoritmo deberá aprender las relaciones entre ellos y dicho proceso no siempre es sencillo. Además, codificando los genes como combinaciones, se reducen las dimensiones del problema y, por lo tanto, se reduce el coste computacional al realizar búsquedas en espacios de altas dimensiones.

- En el caso de la codificación indirecta es necesaria una función de transformación. Esta función puede provocar que distintas codificaciones genotípicas den lugar al mismo fenotipo provocando redundancia en el espacio de búsqueda.

3.1.2 Reseña histórica

El concepto de Computación Evolutiva (CE) no es algo nuevo. Las ideas relacionadas con este campo fueron sugeridas desde los inicios de la era de la computación. Las primeras investigaciones sobre evolución artificial surgieron entre los años 1948 y 1960, cuando Alan Turing comenzó sus estudios sobre las relaciones entre la evolución natural y el aprendizaje [Turing, 1950]. También John von Neumann dedicó sus investigaciones a los autómatas celulares evolutivos que poseían unos mecanismos que les otorgaban un poder computacional equivalente a una máquina de Turing.

En la década de los cincuenta surgen las primeras ideas sobre aplicación de estas técnicas en problemas reales. En 1956, George Friedman formuló una propuesta de aplicación de técnicas evolutivas a la robótica; en 1957, George Box propone su utilización en producción industrial (EVOP - Evolutionary Operation) y en 1958 Friedberg combinó técnicas de aprendizaje por refuerzo y de computación evolutiva para desarrollar instrucciones de un programa.

Los precursores de lo que actualmente conocemos por algoritmo evolutivo fueron Woodrow Bledsoe y Hans Bremermann. En 1960 interpretaron la evolución como un proceso de optimización, sugirieron la codificación binaria, el uso de un valor de aptitud y las estructuras de poblaciones. En 1963, Newell y Simon propusieron el *General Problem Solver* que era capaz de resolver problemas sencillos utilizando evolución, siendo la primera idea de un algoritmo genérico que funcionaba independientemente del problema y dominio de aplicación.

En estos primeros años, destacaron tres grupos de investigación cuya actividad ayudó a definir lo que ahora conocemos como computación evolutiva. En la Universidad de California - Los Ángeles (UCLA) a finales de los sesenta, el grupo formado por Fogel, Owel y Walsh, comenzó a utilizar procesos de evolución simulada para desarrollar máquinas de estados finitos, dando lugar a la programación evolutiva [Fogel et al., 1966]. Un segundo grupo en la Universidad Técnica de Berlin (TUB) en el año 1973, formado por Rechenberg y Schwefel, comenzaron a estudiar cómo los procesos evolutivos podrían utilizarse para resolver problemas complejos con parámetros reales. De estas investigaciones surgieron las estrategias evolutivas [Rechenberg, 1973, Schwefel, 1981]. Por último en el año 1975, en la Universidad de Michigan (UM), John Holland formalizó sus ideas sobre planes reproductivos y adaptativos en lo que actualmente conocemos por algoritmos genéticos [Holland, 1975]. Estos algoritmos eran sistemas adaptativos y robustos capaces de trabajar ante entornos inciertos y dinámicos, y donde dicha característica de auto-adaptación surgía a partir de la realimentación de interactuar con el entorno.

Durante la década de los setenta, la mayor parte de la investigación en este campo se centró en desarrollos teóricos con dos objetivos. En primer lugar, caracterizar el comportamiento de los sistemas evolutivos. En segundo lugar, tratar de comprender cómo estos sistemas pueden ser utilizados para resolver problemas. De esta etapa de estudios teóricos surgen una serie de algoritmos canónicos que han servido de base para el desarrollo de nuevos algoritmos.

La etapa de maduración de las técnicas de CE tuvo lugar durante la década de los ochenta donde se desarrollaron las primeras conferencias sobre este tema y creció enormemente el

número de aplicaciones a problemas reales. Los algoritmos evolutivos desarrollados hasta este momento comenzaron a aplicarse en la optimización de funciones. A partir de los resultados obtenidos de estas aplicaciones y del uso que se comenzó a dar a los mismos surgieron numerosas modificaciones a los algoritmos canónicos que habían sido desarrollados en etapas anteriores. Por ejemplo, se generalizó el uso de las técnicas multi-parentales, en las cuales a partir de μ padres se generaban λ descendientes. También se estudiaron técnicas para mantener la presión selectiva como, por ejemplo, el escalado dinámico de la calidad o las técnicas de reemplazo elitistas. Aunque la mayor parte de las aplicaciones se centraron en el campo de la optimización, los algoritmos evolutivos fueron aplicados en otras áreas y se combinaron con otras técnicas. Así, surgieron los *Learning Classifier Systems* [Holland, 1976], sistemas de aprendizaje máquina que utilizaban algoritmos genéticos como un componente para la generación de las reglas que utilizaban en su ejecución. También comenzaron a aparecer aplicaciones donde se evolucionaban sistemas complejos como redes de neuronas artificiales, colecciones de reglas para la ejecución de tareas y programas desarrollados en código *LISP*.

En la década de los noventa se produjo la unificación de los tres paradigmas principales que habían surgido como proyectos independientes: los Algoritmos Genéticos, las Estrategias Evolutivas y la Programación Evolutiva. Gracias a esta unificación surge el término *Computación Evolutiva* como nombre del área que los engloba. Como resultado de las interacciones se obtuvo una mejor comprensión de las similitudes y diferencias de los tres paradigmas y un cruce de ideas que permitió mejoras en cada uno de los tres campos. También a principios de los noventa aparece una nueva e importante línea de investigación basada en la CE: la Robótica Evolutiva. Algunos investigadores, como Irman Harvey, Phil Husbands, Dave Cliff, Randall Beer o John Gallagher, propusieron el uso de algoritmos evolutivos para automatizar el proceso de diseño de los sistemas robóticos.

Debido a la gran cantidad de trabajos desarrollados en el campo de la CE, a principios del siglo veintiuno se consideraba ya una disciplina madura. Este gran número de trabajos se vio reflejado en la aparición de tres revistas científicas dedicadas exclusivamente a este campo: *Evolutionary Computation* (disponible desde 1993), *IEEE Transactions on Evolutionary Computation* (disponible desde 1997) y *Genetic Programming and Evolvable Machines* (disponible desde el año 2000). También creció el número de congresos y conferencias cuya temática se centraba en estudios teóricos y aplicaciones en el campo de la CE. Destacan: *Genetic and Evolutionary Computation Conference* (GECCO, se celebra desde el año 2001), *Congress of Evolutionary Computation* (CEC, se celebra desde el año 1994), *Parallel Problem Solving from Nature* (PPSN, se celebra desde el año 1990), *Frontiers in Evolutionary Algorithms* (FEA, se celebra desde mediados de los noventa) y *Foundations of Genetic Algorithms* (FOGA, desde el año 1990). El campo de la CE se encuentra en pleno desarrollo en la actualidad y constantemente aparecen nuevas aplicaciones donde comprobar la utilidad de los algoritmos evolutivos. Además, surgen nuevas necesidades que requieren la mejora y la extensión de las capacidades de estos algoritmos, como pueden ser la optimización multi-objetivo, los problemas de optimización con o sin restricciones, sistemas co-evolutivos, sistemas auto-adaptativos, etc.

3.1.3 Paradigmas principales

Tradicionalmente se han considerado cuatro paradigmas principales de algoritmos evolutivos. La filosofía base de estos cuatro paradigmas es la misma y las diferencias entre ellos

radican, generalmente, en el tipo de codificación y en los operadores utilizados en las fases de selección, reproducción y reemplazo. A continuación se explican brevemente estos cuatro paradigmas haciendo hincapié en dichas diferencias.

Algoritmos Genéticos

Como ya se ha mencionado anteriormente, las bases de los algoritmos genéticos (AG) fueron presentadas por John Holland a principios de la década de los sesenta y, más tarde, en 1975, él mismo propuso la idea de los algoritmos genéticos tal y como los conocemos actualmente [Holland, 1975]. El ciclo evolutivo de un algoritmo genético es el mismo que ya ha sido explicado en la sección 3.1.1 y que se describe en el diagrama de la figura 3.2. Cuando John Holland presentó las bases de los AGs hizo especial énfasis en la importancia del desarrollo de un método de propósito general independiente del problema a resolver. Por este motivo, en las primeras versiones de este paradigma se propuso la idea de utilizar un tipo de representación universal para los genes de los cromosomas, la codificación binaria. En versiones posteriores se han utilizado otros tipos de codificaciones, desde números enteros hasta números reales, incluso utilizando combinaciones de distintos tipos de codificaciones.

Una de las principales diferencias de los AGs con respecto a otros paradigmas de AEs es la forma en la que se eligen los individuos que pasarán a la fase de reproducción. En el caso de los AGs, la responsabilidad de decidir qué individuos se reproducen y cuáles no, recae sobre el método de selección y, dependiendo del operador utilizado y de la calidad del individuo, la probabilidad de reproducirse es mayor o menor. Los operadores de selección más utilizados en la bibliografía son: la selección por ruleta, la selección por ruleta con escalado [Hancock, 1994], la selección por posición [Walter, 1953] o la selección por torneo [Goldberg and Deb, 1991].

Otra diferencia importante es el método de reproducción mediante el cual, a partir de los individuos seleccionados de la población de padres, se genera la población de descendientes. En los AGs se le da mayor importancia a la reproducción sexual, en forma de operador de cruce, que a la reproducción asexual, operador de mutación. En el caso de los operadores de cruce se genera la descendencia a partir de la información que proporcionan varios padres, generalmente dos aunque hay casos en los que se utiliza un número mayor de padres. El operador de cruce más utilizado en el caso de los AGs es el cruce por un punto. Existen otras variantes de cruce usando, por ejemplo, dos o tres puntos de cruce en lugar de uno, o también creando descendientes alternando los genes de los padres, llamado cruce uniforme. Los operadores anteriores se desarrollaron para ser utilizados con cualquier tipo de codificación, aunque principalmente con la codificación binaria. Desde el momento en el que los algoritmos genéticos con codificación real (RCGA) cobraron importancia, se desarrollaron operadores de cruce específicos para este tipo de codificación [Herrera et al., 1998, Herrera et al., 2003]. Algunos de ellos, como por ejemplo el operador de cruce *SBX-*α, simulan el comportamiento de un operador de cruce por puntos.

Aunque el operador de mutación, en el caso de los AGs, no tenga la misma importancia que el operador de cruce y la probabilidad de ejecutarse sea baja, este es un mecanismo que sirve para introducir nueva información genética en la población. Este operador es dependiente de la codificación concreta que se utilice en los cromosomas. Si, por ejemplo, se utiliza codificación binaria la mutación consiste en cambiar el valor de un gen con una determinada probabilidad o elegir dos genes e intercambiar sus posiciones. Si la codificación es distinta se amplia la definición de forma que cada gen se cambia, con una determinada probabilidad,

por otro gen válido siguiendo una distribución de probabilidad como, por ejemplo, una distribución uniforme o una distribución normal. Habitualmente la probabilidad de mutación se escoge de tal manera que por término medio se produzca una mutación en cada cromosoma.

El último operador que se aplica en el proceso evolutivo de un AG es el operador de reemplazo. Este operador es el encargado de decidir qué individuos de la población de padres son sustituidos por individuos de la población de hijos y, por lo tanto, no sobreviven en la siguiente generación.

Estrategias Evolutivas

Las estrategias evolutivass (EEs) fueron ideadas por Rechenberg [Rechenberg, 1973] y Schwefel [Schwefel, 1981] basándose en la idea de *evolución de la evolución*. Al igual que en los AGs, cada individuo está representado por un cromosoma pero, además, tiene asociado los parámetros que modelan su evolución (parámetros estratégicos), los cuales también se evolucionan. Esta es la principal característica diferenciadora de este paradigma con respecto a los AGs. Como se explicará más adelante, la programación evolutiva heredó esta característica de las EEs. Otra de las características que diferencian las EEs de los primeros AGs es el tipo de representación. Desde un principio, las EEs fueron desarrolladas para resolver problemas de optimización. Por lo tanto, los genes de los cromosomas se representaban mediante números reales, ya que es la forma natural de representar los parámetros de un problema de optimización.

El ciclo evolutivo de una EE sigue los mismos pasos que el descrito para los AGs, sin embargo, y a diferencia estos, en este caso se le da mayor importancia a la mutación frente al cruce. De hecho, en las primeras propuestas no existía operador de cruce y los descendientes se generaban únicamente a partir de mutaciones de los padres.

En la actualidad, se puede considerar que existen dos tipos de EEs, las cuales se diferencian por los mecanismos de selección de padres y de reemplazo. Estos dos tipos de EEs son las estrategias $(\mu + \lambda)$ o estrategias *plus* y las estrategias (μ, λ) o estrategias *comma*. En el caso de las estrategias $(\mu + \lambda)$, se seleccionan μ padres para generar λ descendientes $(1 \leq \mu \leq \lambda < \infty)$. A la hora del reemplazo se unen los μ padres y los λ descendientes y se eligen los μ mejores individuos que pasarán a la siguiente generación. Esta estrategia es una forma de implementar elitismo, ya que los mejores individuos siempre sobrevivirán generación tras generación. En la segunda estrategia, (μ, λ) donde $1 < \mu < \lambda < \infty$, se seleccionan μ padres para generar λ descendientes y se seleccionan los μ mejores individuos de entre los λ descendientes. En esta estrategia no existe elitismo, por lo tanto presenta poca presión selectiva y mayor diversidad comparada con la estrategia *plus*. La elección de una u otra estrategia depende de las características concretas del problema que se desee resolver.

Como ya se ha mencionado anteriormente, en el caso de las EEs el operador de mutación prima sobre el operador de cruce y aunque el operador de cruce no es muy utilizado en este paradigma, algunos autores recomiendan su uso ya que, en algunas ocasiones, utilizar solamente mutación provoca una rápida convergencia hacia óptimos locales. Rechenberg [Rechenberg, 1973] propuso dos aproximaciones para el operador de cruce que se diferencian principalmente en el número de padres que se utilizan, estas dos aproximaciones son: el cruce local y el cruce. Una vez que se han generado descendientes cruzados se les aplica el operador de mutación con una probabilidad de 1, es decir, todos los genes de todos los individuos serán mutados. Para ejecutar el operador de mutación se ejecutan dos pasos: en primer lugar se auto-adaptan los parámetros estratégicos, como se explicará más adelante, y en segundo lu-

gar se genera el descendiente mutado. El individuo mutado se genera sumándole a cada gen del individuo padre un valor que sigue una distribución de probabilidad determinada por los parámetros estratégicos.

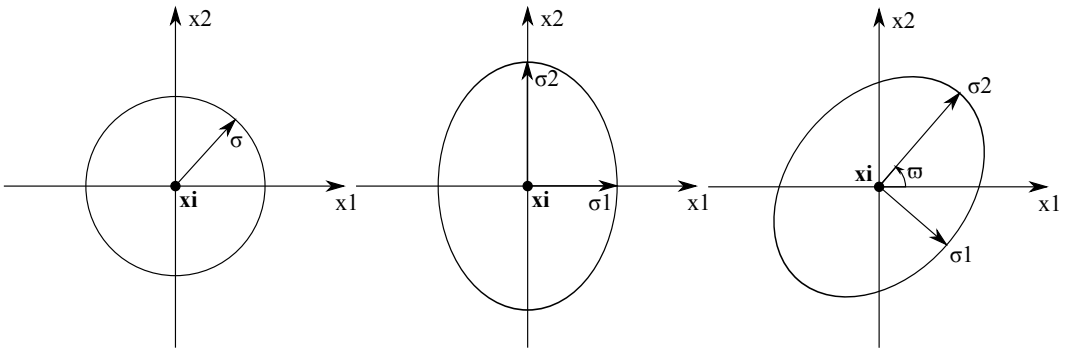
Una de las principales características de las EEs, que ya ha sido mencionada pero merece ser destacada, es el concepto de *evolución de la evolución*, es decir, que los mismos parámetros que gobiernan la evolución también son evolucionados con el objetivo de ajustar la distribución de mutación de forma que el proceso de búsqueda se adapte a la superficie del problema y determine la mejor dirección de búsqueda en la misma. Este proceso de auto-adaptación de parámetros estratégicos se realiza en el primer paso del operador de mutación, previo a la generación de nuevos individuos utilizando dichos parámetros adaptados. Los parámetros estratégicos utilizados en las EEs son la desviación estándar del tamaño del paso de mutación (σ) y los ángulos rotacionales de dichos pasos (ω), estos parámetros pertenecen a una distribución normal. Existen diversas técnicas de auto-adaptación dependiendo del número de parámetros de cada tipo que se utilicen [Bäck, 1996, Hildebrand et al., 1999]. Considerando n_σ el número de desviaciones estándar y n_ω el número de ángulos rotacionales utilizados, las técnicas de auto-adaptación más utilizadas son las que se muestran en la figura 3.3.

Programación Evolutiva

El paradigma de programación evolutiva (PE) fue desarrollado por Fogel [Fogel et al., 1966, Fogel, 1995, Fogel, 1999] y surge a principios de la década de los sesenta. Este paradigma se centra en el desarrollo de modelos de comportamiento (fenotípicos) y no modelos genotípicos. Fogel se basó en sus ideas de que la inteligencia es una propiedad que permite a un sistema adaptar su comportamiento para alcanzar sus metas [Fogel, 2000] con el objetivo de desarrollar un sistema que permitiese la adaptación del comportamiento de los individuos ante un entorno dado.

Los primeros ejemplos fueron desarrollados para la evolución de máquinas de estados finitos (FSM). Estos ejemplos consistían en encontrar un conjunto de comportamientos óptimos para dichas máquinas dentro de un espacio de comportamientos observables. La función de calidad consistía en medir el *error de comportamiento* de un individuo con respecto al entorno donde se ejecutaba.

Al ser utilizado para evolucionar FSMs, una de las principales diferencias con respecto



(a) $n_\sigma = 1, n_\omega = 0$ (b) $n_\sigma = n_x, n_\omega = 0$ (c) $n_\sigma = n_x, n_\omega = n_x(n_x - 1)/2$

Figura 3.3: Distribuciones de mutación para EEs

a otros paradigmas de AEs es la representación de los individuos. Estos se codifican como secuencias ordenadas de acciones que definen un comportamiento. Posteriormente, el uso de la PEse ha ampliado a otro tipo de problemas y ha sido utilizada en optimización en dominios continuos [Fogel et al., 1991, Fogel et al., 1990] donde los individuos se representaban como cadenas de números reales.

El ciclo evolutivo sigue los mismos pasos que otros paradigmas de CE. Durante la fase de inicialización, la población se genera de forma aleatoria utilizando una distribución uniforme tratando de cubrir la mayor área posible del dominio.

Una de las diferencias principales de la PE con respecto a otros paradigmas es la ausencia de operador de cruce durante la fase de reproducción. Por lo tanto, el operador de mutación es el principal operador del paradigma y sobre él recae la responsabilidad de conseguir un balance adecuado entre exploración y explotación. El objetivo principal del mismo es el de introducir variabilidad en la población, generando uno o más descendientes a partir de las modificaciones de un padre. De forma general, el operador de mutación en PE se define según la ecuación:

$$x'_{ij}(t) = x_{ij}(t) + \Delta x_{ij}(t) \tag{3.1}$$

Donde $\Delta x_{ij}(t)$, representa el paso de la mutación y sigue una determinada distribución de probabilidad cuya desviación está determinada por el parámetro estratégico σ_{ij} escalado por una función $\Phi : \mathbb{R} \to \mathbb{R}$. Dependiendo de la función Φ se distinguen tres tipos de estrategias en PE:

- **PE No adaptativa:** donde $\Phi(\sigma) = \sigma$ en cualquier momento de la evolución, es decir, la desviación de la distribución de probabilidad del paso de mutación es estática.

- **PE Dinámica:** donde σ cambia con el tiempo utilizando una función Φ determinística, normalmente en función de la calidad de los individuos.

- **PE Auto-adaptativa:** donde el valor de σ varía dinámicamente y se adapta al proceso de búsqueda. Al igual que ocurre en las EEs, cada variable tiene asociado un parámetro σ cuyo valor se adapta durante el proceso evolutivo a la superficie de búsqueda del problema con las mismas estrategias explicadas en caso de las EEs.

La fase de evaluación también es diferente en el paradigma de PE. Mientras que en otros paradigmas la calidad de los individuos se mide de forma absoluta, es decir, cómo de bien se comporta un individuo con respecto al problema que trata de resolverse, en el caso de la PE la calidad de un individuo se mide de forma relativa al resto de la población. Como ya se ha mencionado anteriormente, la calidad de un individuo en PE mide el *error de comportamiento*. Esta medida sirve para asignar una puntuación a cada individuo cuyo valor depende de la comparación de su comportamiento con respecto al comportamiento de un grupo de individuos de la población seleccionados de forma aleatoria. Al utilizar una medida de error relativa y no absoluta, el proceso de selección de supervivientes se transforma en un proceso competitivo donde padres y descendientes luchan por sobrevivir basándose en su rendimiento frente a un grupo de competidores. Una vez que se calcula la calidad relativa en la fase de evaluación, cada individuo recibe una puntuación basada en dicha calidad relativa. En el proceso de selección se puede utilizar cualquiera de los métodos de evaluación vistos en otros paradigmas: selección por torneo, selección por posición, ruleta, etc., utilizando como valor de calidad la puntuación de cada individuo.

Programación Genética

La programación genética (PG) [Koza, 1992b, Koza, 1994, Koza et al., 1999, Koza et al., 2003] surge a principio de la década de los noventa como una especialización de los AGs. La principal diferencia con respecto a los AGs es la representación de los individuos, ya que en la PG los cromosomas de los individuos representan programas en forma de árboles. En sus orígenes fueron desarrollados para la evolución automática de programas. En cada generación, cada individuo o programa se ejecutaba para medir su rendimiento, cuyo resultado se utiliza como valor de calidad.

La codificación de los individuos en forma de árboles es una de las principales características de este paradigma. Además de ser diferente en cuanto a estructura, el usuario deberá tener en cuenta dos implicaciones importantes de este tipo de representación. La primera, el tamaño de los individuos (medido como profundidad del árbol), estructura (longitud de las ramas y posición de las mismas) y complejidad cambian debido a los operadores de reproducción, y es diferente en cada individuo. La segunda implicación es que es necesaria la definición de una gramática que permita la representación de cualquier posible solución del problema. La gramática de un problema está formada por dos conjuntos: el conjunto de operadores o nodos no hoja y el conjunto de literales o nodos hoja.

En la fase de evaluación, lo más común es presentar al programa generado por el individuo diferentes casos de prueba para medir su rendimiento. Si, por ejemplo, se está desarrollando una expresión binaria, conociendo la salida correspondiente a cada combinación de entradas, la calidad o rendimiento de un individuo se calcula según el número de salidas correctas que prediga la expresión correspondiente. Cuando se desarrollan expresiones matemáticas, la expresión se evalúa calculando el error obtenido por el individuo con respecto al valor objetivo, generalmente los valores objetivo se dan en forma de patrones. La PG también se ha utilizado para evolucionar árboles de decisión, en este caso la calidad de un individuo se corresponde con su precisión a la hora de tomar la decisión correcta. Otra posible aplicación son los juegos de estrategia [Koza, 1990, Koza, 1992a], donde la calidad estaría representada por el número de victorias que obtiene el individuo.

Los operadores de selección utilizados en PG son los mismos que se utilizan en el caso de los AGs, con las mismas ventajas e inconvenientes. Sin embargo, debido a la representación en forma de árbol, ninguno de los operadores de reproducción utilizados en AGs o en EEs sirve para el caso de la PG aunque la filosofía de dichos operadores se mantenga, es decir, mantener el equilibrio entre la exploración y la explotación del espacio de búsqueda. En el caso de los operadores de cruce, las distintas aproximaciones utilizadas se distinguen principalmente en el número de descendientes que se generan a partir, siempre, de dos padres: un descendiente o dos descendientes. Cuando se genera un descendiente, se eligen dos padres (uno actuará como origen y el otro como destino), de cada padre se escoge una rama y la rama del padre destino se sustituye por la rama del padre origen. Si en vez de un descendiente se generan dos, en vez de sustituir ramas lo que se realiza es un intercambio de ramas entre los dos padres. En el caso del operador de mutación, es común implementarlo de forma que se adapte al problema particular que resolverá. Aunque muchos de los operadores utilizados en PG son independientes del problema concreto. En el caso de la PG, no existe operador de reemplazo como tal, es decir, no existe ninguna estrategia que decida qué descendientes sustituyen a los padres, si no que todos los padres serán sustituidos, ya sean mejores o peores que los descendientes.

Los cuatros paradigmas anteriores han servido para sentar las bases de cuatro técnicas ampliamente utilizadas en el campo de la CE. Sin embargo, estos paradigmas solamente indican

aspectos generales de los algoritmos que pertenecen a cada uno de ellos. Dentro de cada uno de estos paradigmas se han implementado algoritmos concretos utilizando diferentes técnicas y operadores.

3.1.4 Áreas de aplicación de los AEs

En esta sección se hará una revisión actualizada de las numerosas áreas en las que se han aplicado los AEs como técnicas para resolver problemas complejos durante los últimos años. La generalidad de este tipo de algoritmos permite que sean aplicados a un gran rango de problemas. En la bibliografía podemos encontrar revisiones de sus principales aplicaciones a problemas reales: minería de datos [Freitas, 2001], aplicaciones de ingeniería [Michalewicz, 1997], problemas de clasificación [Espejo et al., 2010a], tecnologías de la comunicación [Tsujimura et al., 2008], sistemas de logística [Gen and Lin, 2008] o visión artificial [Paulinas and Usinskas, 2007]. Los problemas considerados en estas y otras revisiones pueden agruparse en seis grandes áreas de investigación [De Jong, 2006]: problemas de búsqueda, aprendizaje máquina, programación automática, adaptación, diseño y, sobre todo, optimización.

- Los problemas de búsqueda están muy relacionados con los problemas de optimización, pero las diferencias que existen entre ellos hacen que las técnicas que se siguen para resolverlos sean diferentes. Un problema de optimización trata de maximizar o minimizar un valor objetivo, mientras que un problema de búsqueda trata de encontrar un conjunto de valores del espacio de búsqueda que satisfagan una serie de criterios especificados. Es decir, la función de calidad está formada por un conjunto de sub-funciones que deben satisfacerse y que, por lo tanto, pueden tomar dos valores: 0 o 1, indicando que la sub-función no se soluciona o se soluciona, respectivamente. En este tipo de problemas, generalmente, existe más de una solución que proporciona un valor positivo para el conjunto de sub-funciones consideradas. Elegir una u otra solución depende del usuario y, normalmente, no existe una mejor que otra. También puede darse el caso que no haya ninguna solución que satisfaga el conjunto de sub-funciones. Un ejemplo típico de problemas de búsqueda son los problemas SAT (Satisfability problems) donde, dada una función booleana de N variables sin restricciones en el número de cláusulas, el algoritmo deberá encontrar el conjunto de valores (verdadero o falso) para las N variables que satisfagan todas las clausulas. Este problema ha sido tratado en numerosas ocasiones con distintos tipos de algoritmos evolutivos. En [Gottlieb et al., 2002] los autores hacen una revisión de las soluciones sugeridas en la literatura.

- En el área del Aprendizaje Máquina, los AEs también han sido utilizados frecuentemente. El concepto de aprendizaje se centra en el desarrollo de modelos capaces de generalizar comportamientos a partir de información no estructurada en forma de ejemplos. Una vez que el modelo aprende una serie de ejemplos, su calidad se mide en base al éxito que obtiene ante ejemplos no vistos con anterioridad. Los modelos más utilizados y sobre los que se aplican AEs son, por ejemplo, las redes de neuronas artificiales, en las que se ajustan parámetros como pesos o bias por medio de AEs; o árboles y conjuntos de reglas de decisión generados mediante AEs. Generalmente, los AEs aplicados a Aprendizaje Máquina convergen de manera más lenta que las técnicas propias de los modelos (como puede ser un algoritmo de retropropagación en una red de neuronas), sin embargo, los AEs se han demostrado más efectivos en problemas de Aprendizaje Máquina que impliquen decisiones secuenciales, como pueden ser problemas de navegación o juegos que requieran acciones coordinadas a lo largo del tiempo [Smith,

1983, Grefenstette et al., 1990]. Otro de los problemas de las aplicaciones de Aprendizaje Máquina es el alto coste computacional, por tanto, son deseables técnicas que permitan paralelización como lo son los AEs.

- Cuando Fogel y su equipo de investigación presentaron la programación evolutiva, el objetivo que tenían en mente era el desarrollo de una técnica que les permitiese la generación automática de programas para la obtención de agentes inteligentes. De esta forma se libraban de la tarea tediosa de codificar a mano los posibles comportamientos del agente. Así nacía la idea de programación automática [Fogel et al., 1966]. Desde ese momento, el interés en la evolución de programas ha crecido enormemente y se ha aplicado a gran variedad de lenguajes como por ejemplo *LISP* [Fujiko and Dickinson, 1987, Koza, 1992b] o código ensamblador [Cramer, 1985, Ray, 1994], llegando incluso a utilizarse redes de neuronas artificiales [Harp et al., 1989, de Garis, 1991] o sistemas de reglas [Smith, 1983, De Jong, 1987]. La programación automática es una de las áreas que mayores dificultades presenta y más retos supone en el campo de la CE. Varios aspectos críticos juegan un papel importante. En primer, lugar la representación de los programas, ya sea como reglas, árboles, parámetros, etc.; además de que generalmente se utilizan individuos de longitud variable. La existencia de restricciones, tanto sintácticas como semánticas, también juega un papel crucial, ya que los operadores y la función de calidad deberán estar preparados para trabajar con soluciones que no son factibles en uno o en ambos aspectos. Por último, las funciones de calidad deben considerar aspectos como el consumo de memoria o el consumo de tiempo, además de la ejecución correcta de la tarea.

- Los problemas más típicos tratados con AEs utilizan funciones de calidad que son estáticas. Sin embargo, las funciones de calidad que se tratan de resolver en el caso de los problemas de adaptación son funciones dinámicas que cambian a lo largo del tiempo como las que aparecen en [Branke, 2001, Morrison, 2004]. Los trabajos desarrollados en este campo se han centrado en analizar principalmente los siguientes aspectos:

  - Algoritmos que permitan una continua y eficiente adaptación de las soluciones a un entorno cambiante.
  - Encontrar un balance adecuado entre la calidad de la solución y el coste de adaptación a los cambios.
  - Desarrollar métodos que permitan encontrar soluciones robustas cuya calidad no se vea afectada por cambios en el entorno.

  Además, los desarrolladores o usuarios deberán de prestar atención al concepto de escala de tiempo o cómo de rápido cambia el entorno con respecto al tiempo que tarda en ejecutar una generación un evolutivo. Si el entorno cambia de forma tan rápida que no es posible evaluar la calidad de una generación, entonces se deberá considerar otra alternativa que no sea un AE. En caso contrario, si las transiciones se producen después de largos periodos estáticos, el problema se puede tratar con un AE estándar que en cada transición ejecuta una fase de ajuste de parámetros. Los casos intermedios son los de mayor interés para los AEs, que deberán de mantener una alta diversidad en la población para ser capaces de ajustarse dinámicamente a los cambios en el entorno.

- Otra de las áreas de interés clásicas para los AEs es el campo del diseño automático. Este campo puede dividirse en cuatro categorías principales: la optimización evolutiva de diseños, el diseño evolutivo creativo, arte evolutivo y evolución de formas de

vida artificial [Bentley, 1999]. La optimización evolutiva de diseños es, de las cuatro, la categoría más conocida y, generalmente, es considerada como una clase particular de problemas de optimización. En esta sub-clase de problemas se parametriza el diseño creado y se evolucionan los valores de esos parámetros para obtener un diseño que cumpla con una serie de criterios definidos como, por ejemplo, la maximización de la resistencia del producto o la minimización de costes de fabricación. Al contrario que la optimización evolutiva de diseños, en el caso del diseño evolutivo creativo el objetivo principal es el de generar nuevos diseños a partir de poca o de ninguna información: la evolución se guía por un criterio puramente funcional. Cuando, por el contrario, la evolución se guía por un criterio subjetivo y no funcional es cuando se está hablando de arte evolutivo. Esta categoría explota el proceso de la evolución para crear arte que cambia continuamente de acuerdo con un algoritmo evolutivo. En [Romero and Machado, 2007] se hace una revisión de arte y música evolutiva. Por último, la cuarta categoría se centra en la investigación de la evolución de formas de vida artificial como, por ejemplo, autómatas celulares, formas y disposiciones de neuronas o la evolución de morfologías similares a plantas o animales que realizaron Dawkins [Dawkins, 1986, Dawkins, 1990] y Sims [Sims, 1994b, Sims, 1994a]. El objetivo principal de esta categoría del diseño evolutivo consiste en investigar acerca de los mecanismos de evolución natural para encontrar explicaciones sobre la generación de formas que aparecen en la naturaleza.

- El área de la optimización es, sin lugar a dudas, el área de aplicación que mayor interés suscita. El motivo de este alto interés es la gran cantidad de problemas que existen en el campo de la optimización, tanto en optimización discreta como continua, y para los cuales no existen métodos de resolución analíticos. Dentro de los problemas de optimización el sub-tipo más estudiado es la de optimización de parámetros [Bäck and Schwefel, 1993], es decir, encontrar la combinación de valores para una serie de parámetros que optimizan una función dada. Formalmente, dada una función f de n parámetros $x_1, \ldots, x_n$, la optimización consiste en encontrar los valores de $x_1, \ldots, x_n$ que maximicen (o minimicen) f. Los beneficios de la aplicación de los AEs a problemas de optimización pueden resumirse en los siguientes puntos:

  - Los AEs pueden aplicarse tanto a problemas de parámetros discretos como continuos, e incluso a problemas que combinen parámetros de ambos tipos.
  - En el caso de optimización continua, no es necesario que la función a optimizar sea derivable.
  - En principio, no es necesario ningún conocimiento *a priori* de la superficie a optimizar, aunque como demostraremos en capítulos posteriores este conocimiento aumenta las probabilidades de éxito.
  - Los problemas con ruido no afectan en gran medida a los mecanismos de resolución de los AEs.
  - Son algoritmos fácilmente paralelizables.

  En las primeras etapas de la optimización evolutiva los problemas considerados eran problemas con un solo objetivo y que tenían únicamente restricciones en los límites de los valores de los parámetros. A medida que los AEs fueron utilizados para resolver problemas de aplicaciones reales, los autores comenzaron a incluir mayor conocimiento

del dominio del problema en forma de funciones de restricción o problemas multiobjetivo. La funciones de restricción representan relaciones complejas y no lineales entre los parámetros a optimizar, provocando que existan, dentro del espacio de búsqueda, soluciones no factibles. En [Coello Coello, 2002, Mezura-Montes, 2009] se revisan las técnicas para el manejo de estas funciones de restricción, entre las que se incluyen el uso de funciones de calidad que penalizan a las soluciones no factibles u operadores que no permiten generar soluciones que se encuentren dentro de dichas áreas restringidas.

Los problemas multiobjetivo son aquellos en los que existe más de una función a optimizar y en los que, generalmente, no se obtiene una única solución, sino una colección de soluciones que representan compromisos entre los M objetivos que se optimizan. Una forma estándar de tratar estos problemas es generar una única función de calidad ponderando cada uno de los sub-objetivos con un peso. De esta forma se obtiene una de las múltiples soluciones pudiendo aplicar cualquier AE estándar. Sin embargo, si no es posible conocer *a priori* los pesos de cada uno de los sub-objetivos es más recomendable utilizar algoritmos desarrollados específicamente para este tipo de problemas. Desde finales de los noventa existen multitud de alternativas. En [Coello Coello, 1999] podemos encontrar una revisión de las mismas con sus ventajas y desventajas.

La principal conclusión que se debe extraer de esta sección es que el número y variedad de campos de aplicación de los AEs ha aumentado mucho en las dos últimas décadas. Además son cada vez más complejos, dando lugar a mejoras y modificaciones en los algoritmos existentes e incluso a la aparición de otros nuevos como veremos en la siguiente subsección.

3.1.5 Nuevos modelos de AEs

Debido a que este trabajo se centra en el campo de la optimización, la programación genética y la programación evolutiva están fuera del alcance del mismo. Se recomienda [Espejo et al., 2010b] para conocer el estado del arte de dichos campos y su evolución hasta el día de hoy. A partir de este punto nos centraremos en los algoritmos genéticos y las estrategias evolutivas. Concretamente, en esta subsección revisaremos las principales variantes que han surgido en los últimos años y que hemos resumido en cuatro principales: la evolución diferencial, los modelos probabilísticos, los algoritmos culturales y los algoritmos meméticos.

Evolución Diferencial

El concepto de Evolución Diferencial, o Differential Evolution (DE) fue presentado en 1995 por Storn y Price [Storn and Price, 1995]. Comparte muchas características con otros paradigmas y modelos de AEs, sin embargo, se diferencia significativamente de todos ellos en el sentido de que la información sobre distancia y dirección de la población se utiliza para guiar el proceso de búsqueda. Esta es la mayor aportación del concepto de los algoritmos de DE a los AEs estándar.

El DE se considera un tipo de EEs, sin embargo, a diferencia de estas, la longitud del paso de mutación no se obtiene a partir de una distribución de probabilidad parametrizada por el usuario, si no que esta depende de la distribución de la población sobre el espacio de calidad. La posición de los individuos de la población en el espacio de calidad proporciona información sobre dicho espacio. Si la inicialización de la población se realiza de manera uniforme, esta proporcionará una buena representación del espacio de búsqueda, con grandes distancias entre individuos. A medida que progresa el proceso de búsqueda, las distancias

entre individuos se hacen cada vez más pequeñas, haciendo que la población converja hacia una solución. Las distancias entre los individuos son una buena indicación del grado de diversidad de la población y del orden de magnitud de la longitud del paso de mutación que debería aplicarse para que la población converja hacía un punto:

- Si la distancia entre los individuos es grande, estos harán saltos grandes durante la evolución y explorarán el mayor espacio posible.
- Si las distancias son cortas, los pasos serán pequeños y la tendencia será explotar áreas pequeñas.

El comportamiento del DE se basa en estas dos premisas. En las primeras generaciones las distancias son grandes y el algoritmo tiende a explorar el espacio de búsqueda, a medida que pasan las generaciones la población converge hacia zonas prometedoras, las distancias se acortan y el algoritmo tiende a explotar dichas zonas prometedoras. Este comportamiento se obtiene calculando los pasos de mutación como una diferencia ponderada de varios individuos. Se utilizan los llamados vectores diferenciales para determinar la magnitud y la dirección en los pasos de la mutación.

La principales ventajas de este modelo con respecto a otros modelos de AEs son:

1. La información sobre la superficie de búsqueda, representada en la población actual, se utiliza para dirigir la búsqueda.
2. De acuerdo con el teorema central del límite [Davidson and Davidson, 1994], cuando la población es lo suficientemente grande como para permitir un buen número de vectores diferenciales, la distribución que gobierna la longitud de los pasos de la mutación tiende hacia una distribución normal de media cero [Storn, 1996b]. Esto, unido al hecho de que los individuos que generan la descendencia se seleccionan siguiendo una distribución de probabilidad uniforme, evita que se produzca el efecto de *deriva genética* y, como consecuencia, una falta de variabilidad en la población.

El DE ha sido ampliamente utilizado no solo en el campo de la optimización de funciones de codificación real [Price, 1997, Storn, 1996b, Storn and Price, 1997], sino también, debido sobre todo a su sencillez y buenos resultados, a otros problemas entre los que se incluyen el entrenamiento de redes neuronales [Abbass, 2002, Chen et al., 2002, Magoulas et al., 2001, Magoulas et al., 2004], problemas de clasificación [Omran et al., 2005, Paterlini and Krink, 2004], evolución de controladores [Lopez Cruz et al., 2003a, Lopez Cruz et al., 2003b, Wu and Handroos, 2000, Joshi and Sanderson, 1997], diversos problemas de diseño como diseño de filtros [Chang and Chang, 1998, Storn, 1996a] o diseño de sistemas [Babu and Sastry, 1999, Kurup et al., 2003, Kyprianou et al., 2001], aplicaciones para el análisis de imágenes [Li et al., 2003, Omran et al., 2005] o problemas de planificación [Rae and Parameswaran, 1998, Rzadca and Franciszek, 2005].

Modelos probabilísticos

Los modelos probabilísticos también llamados algoritmos de estimación de distribuciones (EDA) son un tipo de algoritmos evolutivos en los que durante la fase de reproducción su comportamiento no se basa en los modelos genéticos de cruce y mutación tal y como los conocemos, sino que simulan el comportamiento evolutivo mediante la estimación de una distribución de probabilidad para generar nuevos individuos. Esta estimación de la distribución

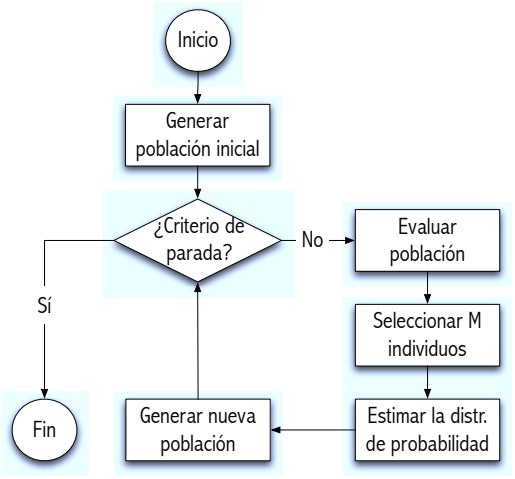


Figura 3.4: Diagrama de flujo de un algoritmo EDA estándar.

de probabilidad puede verse como una generalización del operador de cruce multiparental ya que se utiliza información de más de dos padres, el operador de selección en este caso consiste en seleccionar qué individuos aportarán su información a dicha distribución de probabilidad. El uso de un modelo de probabilidad sustituyendo a los operadores de reproducción proporciona a los EDAs dos ventajas importantes frente a los AEs clásicos. En primer lugar, la ausencia de parámetros de configuración como, por ejemplo, las probabilidades de mutación y cruce. En segundo lugar, el uso de un modelo probabilístico que guía el proceso de búsqueda proporciona mayor claridad al algoritmo.

En la figura 3.4 se muestra un diagrama de flujo que representa el proceso que siguen los EDAs a la hora de resolver un problema. Los pasos que sigue la ejecución de un algoritmo de este tipo son los siguientes:

1. Generar la población inicial de N individuos de forma aleatoria.
2. Mientras que no se cumpla el criterio de parada:
   - 2.1. Evaluar los individuos frente a la función de calidad utilizada.
   - 2.2. Seleccionar M individuos de la población ($M \leq N$) de acuerdo con el operador de selección utilizados.
   - 2.3. Estimar la distribución de probabilidad de los individuos seleccionados.
   - 2.4. Generar N individuos utilizando la distribución de probabilidad estimada.

El comportamiento de los EDAs se basa en que es posible realizar un modelo probabilístico de las áreas más prometedoras de un espacio de búsqueda y utilizar dicho modelo para guiar la búsqueda hacia el óptimo. Este modelo se consigue construyendo una distribución probabilística que permite realizar una estimación de las características que comparten los puntos seleccionados. De esta forma es posible capturar diferentes patrones de interacción entre subconjuntos de variables del problema y utilizar este conocimiento para generar nuevas soluciones. De esta forma los EDAs son capaces de resolver problemas donde existen fuertes interacciones entre componentes del problema, mientras que otros AEs, como los AGs o las EEs, no son capaces de extraer información de estas interacciones.

En la bibliografía se pueden encontrar multitud de implementaciones de EDAs. La elección de una u otra para la resolución de un problema determinado no es una tarea sencilla.

Los modelos más simples generalmente tienen poco coste computacional pero son bastante limitados a la hora de representar interacciones de orden alto entre variables. Por otro lado, los modelos más complejos, que sí son capaces de representar relaciones de mayor complejidad, requieren, para su implementación, estructuras de datos más sofisticadas y su coste computacional es mayor.

Una posible clasificación de algoritmos EDAs puede realizarse teniendo en cuenta dos características [Armananzas et al., 2008]: su capacidad para aprender las dependencias entre variables y cómo se aprende el modelo probabilístico. De acuerdo con su capacidad para aprender dependencias entre variables se distinguen tres tipos de EDAs:

- **Modelos uni-variables:** este tipo de modelos asume que todas las variables del problema son independientes, por lo tanto, son los algoritmos más simples dentro de los EDAs. Ejemplos de este tipo de algoritmo son el algoritmo *Population-based incremental learning* (PBIL) [Baluja, 1994], *Compact Genetic Algorithm* (cGA) [Harik et al., 1999] y *Univariate Marginal Distribution Algorithm* (UMDA) [Mühlenbein and Paasz, 1996].

- **Modelos bi-variables:** permiten representar dependencias de orden bajo entre variables. Los algoritmos *Mutual-Information-Maximizing Input Clustering* (MIMIC) [De Bonet et al., 1997], *Bivariate Marginal Distribution Algorithm* (BMDA) [Pelikan and Muhlenbein, 1999], *Dependency Tree-Based EDA* [Baluja and Davies, 1997] y Tree-EDA [Santana et al., 1999] son algunos ejemplos.

- **Modelos multi-variables:** este tipo de EDA utiliza modelos probabilísticos de orden mayor que dos, por lo tanto permiten representar relaciones complejas entre variables del problema. Algunos ejemplos de este tipo de algorimos son: *Factorized Distribution Algorithm* (FDA) [Mühlenbein et al., 1999], *Estimation of Bayesian Network Algorithm* (EBNA) [Etxeberria, R. and Larrañaga, P., 1999], *Bayesian Optimization Algorithm* (BOA) [Pelikan et al., 2000] o su extensión *Hierarchical BOA* (hBOA) [Pelikan and Goldberg, 2006] y, por último, *Extended Compact Genetic Algorithm* (EcGA) [Harik et al., 1999].

Teniendo en cuenta el método de aprendizaje que se utiliza para aprender el modelo probabilístico se distingue entre:

- **Aprendizaje de parámetros:** en este tipo de algoritmos EDA se utiliza un modelo probabilístico especificado al inicio de la evolución y del cual únicamente se ajustan los parámetros que definen tal modelo. Los algoritmos PBIL, cGA, UMDA y FDA se incluyen dentro de este tipo de algoritmos.

- **Aprendizaje de estructura y parámetros:** en este segundo tipo además de los parámetros del modelo el algoritmo deberá estimar también su estructura. De los algoritmos vistos en la clasificación anterior MIMIC, EcGA y los EDAs que utilizan modelos Bayesianos o Gaussianos pertenecen a este tipo.

Como otros modelos de AEs, la principal aplicación de los EDAs se centra en el campo de la optimización. Esta técnica probabilística ha demostrado ser robusta y eficaz tanto en problemas de optimización en dominios discretos [Blanco and Lozano, 2001, Larrañaga et al., 2003] como continuos [Sun et al., 2005] e incluso ha sido aplicado a dominios mixtos combinando variables discretas y continuas [Ocenasek and Schwarz, 2002]. También se han utilizado EDAs para abordar la resolución de problemas típicos de optimización como son el

problema de la mochila [Wu et al., 2010], el problema del viajante [Tsutsui, 2002], la planificación de tareas [Li and Yong, 2009] o problemas de grafos [Bengoetxea et al., 2001]. Los resultados de la aplicación de EDAs a problemas de aprendizaje máquina también han sido satisfactorios. Entre esas aplicaciones se encuentran problemas de selección y ponderación de características [Cantú-Paz, 2002, Inza and Sierra, 2000], inducción de reglas [Sierra et al., 2001], problemas de clasificación [Roure et al., 2001] o ajuste de pesos en redes de neuronas artificiales [Cotta et al., 2001].

Algoritmos culturales

Los AEs estándar ejecutan un proceso de búsqueda denominado imparcial (del inglés, *unbiased*), es decir, no utilizan la información que puede obtenerse del entorno en el que se realiza dicha búsqueda. El conocimiento que se obtiene de este entorno puede servir para guiar el proceso evolutivo hacia zonas prometedores y desviarse de zonas donde el evolutivo no haya proporcionado buenos resultados. Los algoritmos culturales [Reynolds, 1999] tienen en cuenta esta premisa y durante el proceso de búsqueda realizan un tipo de evolución que les permite adaptarse al entorno en el que se trabaja.

Estos algoritmos parten de la definición del concepto de cultura. En el campo de la CE, el concepto de cultura se define como una fuente de información que tiene influencia sobre el comportamiento de los individuos de la población. Esta información cultural se encuentra accesible para todos los individuos a lo largo de todas las generaciones [Engelbrecht, 2007]. En cambio, en los AEs estándar, el conocimiento que se posee sobre el entorno no se conserva generación tras generación.

Para la implementación de un algoritmo cultural son necesarios dos espacios de búsqueda: un espacio de búsqueda poblacional, el mismo espacio de poblaciones que utilizan los AEs estándar; y un espacio de creencias, el cual representa el conocimiento o cultura adquirida a lo largo de las generaciones. Ambos espacios evolucionan en paralelo y ambos influyen uno sobre el otro. El espacio de creencias es un elemento clave para los algoritmos culturales, representa un repositorio de conocimiento donde se almacena el comportamiento colectivo de los individuos de la población. En este espacio se almacena información sobre generalizaciones en las experiencias de los individuos, considerando experiencia como la información que obtienen durante el proceso de evolución en forma de calidad de diferentes zonas del espacio de búsqueda. Las experiencias se almacenan y se modelan a lo largo de las generaciones, no únicamente durante una generación. Estas generalizaciones pretenden representar las creencias sobre cuál es el comportamiento óptimo de los individuos.

Además de los dos espacios de búsqueda para la implementación de un algoritmo cultural es necesario un protocolo de comunicaciones entre ambos espacios. Este protocolo de comunicaciones tiene dos canales y cada uno de ellos implementa un comportamiento diferente: un canal cuyo comportamiento es el de seleccionar qué individuos del espacio de búsqueda servirán para adaptar el conjunto de creencias y un canal cuyo comportamiento es el de decidir cómo las creencias adquiridas influyen en los individuos de la población.

La función de aceptación es otro de los componentes importantes de un algoritmo cultural y determina qué individuos de la población actual serán utilizados para modelar las creencias. Para implementar este tipo de selección se puede utilizar cualquier mecanismo de selección de los AEs estándar, tanto operadores de selección estáticos como dinámicos.

El esquema general de un algoritmo cultural está representado en la figura 3.5. El funcionamiento básico de un algoritmo cultural sigue los pasos que se describen a continuación:

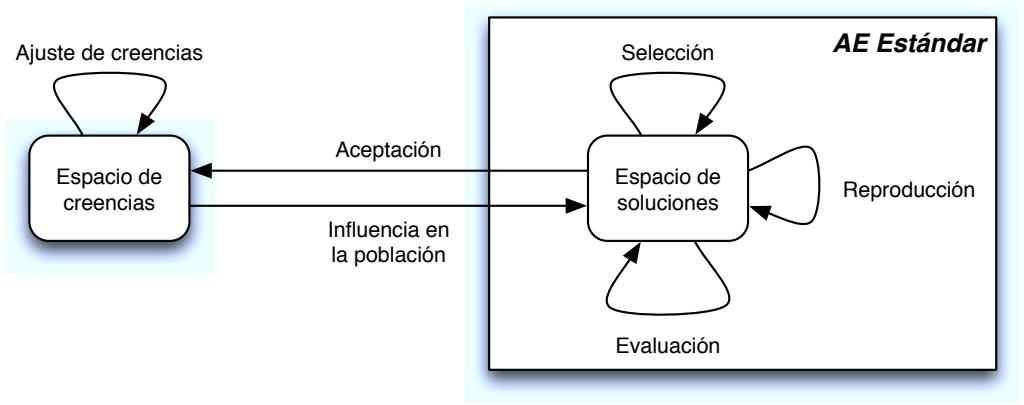


Figura 3.5: Esquema básico de un algoritmo cultural estándar

1. Los individuos del espacio de búsqueda se evalúan con la función objetivo del problema a resolver.

2. La función de aceptación determina qué individuos de la generación actual tendrán influencia sobre la información almacenada en el espacio de creencias.

3. Las experiencia representada en las soluciones aceptadas adaptan el espacio de creencias de la generación actual.

4. Las creencias adaptadas se utilizan para adaptar el proceso de evolución. Esta adaptación se realiza modificando los parámetros de los operadores del evolutivo base.

5. La población del espacio de soluciones se evoluciona con un AE estándar adaptado por la información del espacio de creencias.

Los algoritmos culturales han sido aplicados para la resolución de numerosos problemas de diferentes campos como por ejemplo: el modelado de la evolución de la agricultura en el valle de Oaxaca (México) [Reynolds, 1979, Reynolds and Peng, 2004], aplicaciones de aprendizaje conceptual [Sverdlik et al., 1992], inducción de arboles de decisión [Reynolds and Al-Shehri, 1998], optimización de funciones de parámetros reales [Reynolds and Chung, 1997], optimización de redes semánticas [Rychtychyj and Reynolds, 1999], validación de software [Ostrowski and Reynolds, 1999], minería de datos [Jin and Reynolds, 2000], segmentación de imágenes [Reynolds and Rolnick, 1995] o robótica [Franklin and Bergerman, 2000].

Algoritmos meméticos

Los algoritmos meméticos [Moscato, 1989] engloban una serie de metaheurísticas cuyo desarrollo comenzó a principio de los 80 y cuya principal característica es la hibridación de diferentes algoritmos. En el campo de los algoritmos meméticos se ha hecho especial hincapié en la hibridación utilizando como base algoritmos poblacionales, como son los AEs [Hart et al., 2004]. También son relevantes los algoritmos meméticos basados en técnicas de recocido simulado [Fidanova et al., 2009] o búsqueda tabú [Glover and Taillard, 1993]. Otra de las características importantes de este tipo de metaheurísticas es que, con el objetivo de acelerar el proceso de optimización, utilizan para dicho proceso conocimiento dependiente del

problema. El término *memético* proviene del concepto *meme* presentado por Dawkins [Dawkins, 1990] y se utiliza también en los algoritmos culturales. Un *meme* representa una unidad de conocimiento o de información que proporciona ideas, conceptos, modas y tradiciones de una sociedad. Los algoritmos meméticos, además de hibridar diferentes técnicas de búsqueda, tratan de incorporar conocimiento del problema al proceso de búsqueda a través de *memes* que pueden ser útiles para hallar de forma eficiente soluciones mejores. En el área de la CE, los algoritmos meméticos suelen recibir otros nombres como AEs híbridos o AEs Lamarckianos.

Al contrario que los AEs tradicionales, los algoritmos meméticos han sido desarrollados para explotar la mayor cantidad posible de conocimiento sobre el problema a resolver. En el campo de los AEs, las ventajas de este tipo de técnicas fueron rechazadas durante muchos años a pesar de los trabajos que defendían las técnicas de hibrización para mejorar los resultados de los AEs [Davis, 1991]. Fue con la aparición del *No-Free-Lunch Theorem* (NFL) [Wolpert and Macready, 1997], del cual es posible concluir que el rendimiento de los algoritmos de búsqueda depende de la cantidad y la calidad del conocimiento del dominio que se incorpore, cuando comenzaron a cobrar mayor importancia [Moscato and Cotta, 2003]. Para explotar el conocimiento del dominio los algoritmos meméticos incorporan heurísticas previamente desarrolladas como por ejemplo: pre-procesado de la información, técnicas de búsqueda local, operadores de reproducción especializados, etc.

Combinando diferentes técnicas de búsqueda en un solo algoritmo se potencia el mecanismo cooperativo de los AEs tradicionales incorporando mecanismos guiados de competición. Los individuos, que en la terminología de los algoritmos meméticos reciben el nombre de *agentes*, poseen capacidades para intentar mejorar sus soluciones y compiten entre sí para propagar sus *memes*. El funcionamiento típico de un algoritmo memético es el de un AE tradicional al que se le incorpora un mecanismo de búsqueda local basado en los *memes* del individuo. Para realizar este proceso existen dos alternativas:

- Incorporar un nuevo operador de búsqueda que utilice la información de los *memes*. En este caso se utilizan meta-operadores que iteran sobre los resultados de aplicar los operadores de cruce y mutación tradicionales. Estos meta-operadores pueden ser, por ejemplo, un operador de búsqueda local que itere un cierto número de pasos sobre una solución potencial al problema.

- Incluir la información memética en los operadores típicos de cruce y mutación. En este caso hay que modificar los operadores incluyendo mecanismos que exploten la información del problema.

Además de en los operadores, en los algoritmos meméticos el conocimiento que se extrae del problema se aplica en la fase de generación de la población. En este paso se puede generar la población inicial aleatoriamente, como en los AEs tradicionales, o incorporar conocimiento. Si se opta por incorporar conocimiento en esta fase, la población inicial puede generarse empleando soluciones proporcionadas por una heurística existente o generar una población aleatoria a la que se le aplican optimizadores locales durante una serie de pasos antes de comenzar el proceso evolutivo. Además, puede incorporarse un mecanismo de reinicio de la población, este mecanismo se aplica generalmente cuando la población de agentes converge y no es posible obtener mejoras en la población.

Los buenos resultados de los algoritmos meméticos en problemas NP-completos, ha hecho que se conviertan en algoritmos muy populares a la hora de resolver problemas del ámbito de la optimización combinatoria. Podemos encontrar aplicaciones de algoritmos meméticos

en problemas de particionado de grafos [Merz and Freisleben, 2000, Yeh, 2000], empaquetado [Reeves, 1993], coloreado de grafos [Coll et al., 1999, Costa et al., 1995], planificación de tareas y fechas de entrega [Franca et al., 1999], problemas de asignación [Chu and Beasley, 1997] o problemas del viajante (TSP) [Moscato and Norman, 1992]. Además de estos problemas muy conocidos en el mundo de la optimización, los algoritmos meméticos han sido utilizados con el mismo éxito en otros problemas de optimización combinatoria menos conocidos, como el diseño de trayectorias óptimas para naves espaciales [Crain et al., 1999] o el emparejamiento parcial de formas geométricas [Ozcan and Mohan, 1998]. Destacan también aplicaciones en el campo del aprendizaje máquina y la robótica donde han sido utilizados para el entrenamiento de redes neuronales [Ichimura and Kuriyama, 1998], reconocimiento y clasificación de características [Aguilar and Colmenares, 1998], aprendizaje de comportamientos reactivos en agentes móviles [Cotta and Troya, 2000] o el análisis de series temporales [Östermark, 1999].

Como conclusión de esta sección centrada en una revisión actualizada del campo de los AEs, diremos que se han descrito los cuatro paradigmas clásicos de la computación evolutiva y se han enfatizado las numerosas áreas de aplicación de los mismos, incluyendo problemas de búsqueda, aprendizaje máquina, programación evolutiva, problemas adaptativos, diseño automático y optimización. Como consecuencia, esto ha dado lugar a la aparición de nuevas mejoras y nuevos algoritmos para afrontar de forma eficiente dichas aplicaciones. Centrándonos en los problemas de optimización, para el caso de los algoritmos genéticos y las estrategias evolutivas, se han presentado las cuatro principales líneas de desarrollo que se han seguido en los últimos años y que continuarán en el futuro.

La siguiente sección estará dedicada a mostrar el principal problema derivado de este auge en la aplicación práctica y el desarrollo de los AEs en un periodo tan corto de tiempo: la falta de estandarización y de formalización en los algoritmos que se presentan.

3.2 Ausencia de caracterización formal: evidencias y consecuencias

Debido al elevado número de AEs que han ido surgiendo en la última década y a las numerosas modificaciones sobre dichos algoritmos, es común encontrar en las revistas especializadas trabajos dedicados al análisis y comparación de los mismos. A pesar de esto, el campo de la computación evolutiva sigue careciendo de una metodología estándar que sigan todos los desarrolladores y que permita caracterizar formalmente los algoritmos a partir de los resultados presentados en dichos trabajos. A lo largo de esta sección se presentarán evidencias sobre los principales problemas detectados a nivel de análisis formal de algoritmos y se realizará una revisión bibliográfica de los trabajos centrados en dicho análisis con el objetivo de mostrar qué esfuerzos se han llevado a cabo tratando de solucionar la falta de estandarización del campo y qué falta por hacer. Concretando, en una primera subsección se mostrará la metodología seguida por los usuarios de AEs a la hora de seleccionar y aplicar dichos algoritmos, y en una segunda se analizará la metodología seguida por los diseñadores a la hora de publicar sus desarrollos.

3.2.1 Metodología seguida por los usuarios

Una de las principales consecuencias de la ausencia de una caracterización formal y estándar a la hora de presentar AEs es también la ausencia de formalización en los trabajos en los cuales se utilizan estas técnicas aplicados en casos reales tales como problemas de ingeniería, de diseño automático, de planificación, etc. Por ejemplo, podemos encontrar las siguientes evidencias:

- En muchos de los trabajos presentados se utilizan algoritmos modificados "ad-hoc" para el problema concreto, sin tener en cuenta su generalización a otras clases de problemas similares. Por ejemplo, en [Salcedo-Sanz et al., 2010] los autores utilizan un AG modificado para permitir el uso de la codificación Dandelion [Thompson et al., 2007] común para codificar árboles. Los operadores desarrollados para el algoritmo son totalmente dependientes del problema y de la codificación. Los resultados obtenidos se comparan con otro algoritmo utilizado para el mismo problema y la aproximación presentada mejora estos resultados, pero no alcanza los valores considerados como soluciones óptimas. También en [Tarzjan and Moghaddam, 2007] se utiliza una variación de un AG, en este caso para solucionar un problema de indexado de imágenes. Los autores hacen una revisión de otras técnicas utilizadas para resolver este problema, todas ellas basadas en AGs. El mayor problema de esta aplicación está en el gran coste computacional que implica el indexado de gran número de ficheros. A pesar de que los AGs no son los AEs con una mayor velocidad de convergencia, los autores no se plantean analizar otro tipo de AE e insisten en modificar un AG. Lo mismo ocurre en [Juang et al., 2008], donde se trata de aplicar un AG a un Proporcional Integral Derivativo (PID). En la introducción de dicho trabajo se revisan otras aproximaciones donde también se han aplicado variaciones de un AG. Los autores implementan una nueva modificación de un AG mejorando los resultados anteriores pero, de nuevo, no se plantea la generalización del algoritmo a problemas del mismo tipo.

- Los resultados que se obtienen son sólo para instancias concretas de un problema. Normalmente, si se modifica algún parámetro de esa instancia la configuración del algoritmo utilizado no es válida y, por lo tanto, tampoco lo son los resultados obtenidos. Por ejemplo, justificar que un DE es bueno para solucionar problemas de planificación basándose en trabajos anteriores puede no ser correcto, ya que las instancias que se resuelven solamente comparten la característica de ser problemas de planificación [Qian et al., 2008]. La dimensión del problema, el tipo concreto de problema de planificación, etc. son características de la aplicación que pueden afectar al funcionamiento de un AE. Existen AEs que funcionan mejor en problemas de pocas dimensiones pero cuyo comportamiento se degrada a medida que estas crecen. Sin embargo, otros son más escalables en cuanto a dimensiones y su comportamiento se degrada en menor medida. Además, la tónica general de los trabajos de aplicación presentados es la de no justificar la elección de los parámetros de configuración utilizados más allá del hecho de que han sido los que mejores resultados han proporcionado.

- Son numerosos los trabajos donde no se analizan los problemas a priori para tomar la decisión sobre qué algoritmo utilizar. Es decir, en ningún momento se realiza un análisis previo para descartar de antemano alguno de los algoritmos y de esta forma ahorrar tiempo en los experimentos. Existen casos en los cuales se analizan los problemas pero las conclusiones que se obtienen no se utilizan para la elección de un algoritmo. Por

ejemplo, en [Schneider et al., 2005] los autores utilizan una EE para entrenar los parámetros de una red de neuronas artificiales en un problema de clasificación de imágenes. En este trabajo se comparan dos posibles codificaciones para los cromosomas: una codificación directa y una indirecta. Los parámetros que se evolucionan se han dividido en dos tipos. Por un lado, los parámetros propios de la red y, por otro lado, unos pesos que determinan cómo se combinan las características de las imágenes. Se conocen las relaciones no lineales que existen entre los parámetros de la red, sin embargo, los autores no plantean otra posible solución además de utilizar una EE con dos pasos de mutación diferentes: uno para los parámetros de la red y otro para los pesos de combinación. Conociendo *a priori* que existen relaciones entre los parámetros, quizás la mejor opción no es utilizar una EE simple ya que existen otras aproximaciones [Auger and Hansen, 2005, Goldberg et al., 1989] que permiten al algoritmo estimar esas relaciones y aprovecharlas durante el proceso evolutivo para facilitar la búsqueda.

- Cada vez se presentan más algoritmos híbridos intentando sacar el mejor partido de cada uno de los algoritmos particulares utilizados en la hibridación. Esto se demuestra en el crecimiento del uso de algoritmos meméticos. Por ejemplo, en [Yang et al., 2011] los autores integran un AG, un algoritmo *potential path search* y un algoritmo de simulación del problema concreto para tratar de resolver el problema de planificación del trasporte de carga en ferrocarril. Los resultados obtenidos mediante esta técnica híbrida no se comparan con ninguna otra aproximación. Los autores indican que será trabajo futuro y no se ha considerado otro AE como base del algoritmo híbrido desarrollado.

Los ejemplos anteriores son ilustrativos del problema derivado de la falta de una caraterización formal de los AEs. Este problema también existe dentro del Grupo Integrado de Ingeniería (GII), en el cual se ha desarrollado este trabajo, además de en muchos otros grupos de investigación en la actualidad. Este hecho ha constituido el origen motivacional para el desarrollo de este trabajo.

Los trabajos comentados son una muestra del procedimiento utilizado actualmente por los investigadores. Realmente no existe una metodología estándar que facilite la presentación y el posterior análisis de los resultados obtenidos en problemas de aplicaciones reales. Los autores de cada uno de los trabajos conocen los puntos clave de su desarrollo y, en ocasiones, también conocen las características concretas de su problema, pero estas no son utilizadas para justificar la elección de un algoritmo porque en los trabajos en los cuales se presentan algoritmos no se especifica claramente para qué tipo de problemas son más adecuados. No se aprovechan desarrollos previos para mejorarlos basándose en los resultados obtenidos. Simplemente se citan para comparar los nuevos algoritmos y los nuevos resultados, siempre mejores, con los anteriores. En el apartado siguiente se comentarán las críticas que investigadores como Hooker o Eiben han realizado sobre este procedimiento y que, a día de hoy, todavía no han servido para mejorar los trabajos presentados.

Una vez constatado el problema que origina a nivel de usuario la falta de estandarización en los AEs, en la siguiente subsección se analizará el problema desde el punto de vista del diseñador de algoritmos.

3.2.2 Metodología seguida por los diseñadores

El campo de la CE es relativamente joven si lo comparamos con otros también dedicados al desarrollo de algoritmos y técnicas de resolución de problemas de optimización. Sin em-

bargo, a pesar de su juventud y gracias sobre todo al rápido crecimiento de las capacidades computacionales, el desarrollo de nuevos algoritmos es cada vez más frecuente. Como ya se ha comentado, constantemente aparecen nuevos paradigmas, técnicas revisadas y modificaciones sobre algoritmos anteriores que justifican sus buenos resultados comparándose con antiguas versiones. Las competiciones de algoritmos evolutivos son cada vez más numerosas en los congresos y conferencias del área. Véase, por ejemplo, el caso del congreso de la IEEE *Congress of Evolutionary Computation* (CEC) donde, desde el año 2005, se realizan competiciones de AEs incluyendo todo tipo de problemas de optimización. Si bien no deja de crecer el interés por obtener el "mejor" AE, el desarrollo de una metodología adecuada para el análisis y comparación de los mismos sigue siendo una asignatura pendiente [Bartz-Beielstein and Preuss, 2010].

En el campo de la optimización es conocido el *No-Free-Lunch Theorem* (NFL) [Wolpert et al., 1995, Wolpert and Macready, 1997], que puede aplicarse al campo de la optimización con AEs. Este teorema postula que todos los algoritmos de optimización tienen un rendimiento equivalente a la media en el conjunto $\mathscr{F}$ de funciones de optimización de $\mathscr{X}$ en $\mathscr{Y}$, siendo $\mathscr{X}$ e $\mathscr{Y}$ conjuntos finitos. Como consecuencia, si existe un algoritmo A cuyo rendimiento supera a otro algoritmo B en un subconjunto de funciones de $\mathscr{F}$, entonces hay otro subconjunto de funciones de $\mathscr{F}$ donde el rendimiento del algoritmo A es peor que el rendimiento del algoritmo B. Además de ser utilizado como fundamento en contra de la utilización de algoritmos de optimización continua cuando son aplicados con muy poco conocimiento del dominio del problema, el NFL refuerza la idea de que el desarrollo de un algoritmo de optimización adecuado requiere conocer la clase de problema que se está optimizando. Resulta curioso como, a pesar de la gran aceptación del teorema, los investigadores siguen tratando de obtener el "mejor" AE en todo tipo de funciones, de forma que sea lo más general posible y que pueda aplicarse a un amplio rango de problemas.

Si se analizan los trabajos en los cuales se presentan o comparan AEs [Storn and Price, 1995, Hansen and Ostermeier, 2001], a pesar de la falta de estandarización a la hora de presentar los resultados, estos tienden a seguir una estructura similar que puede resumirse en los siguientes apartados:

1. Si se trata de un trabajo en el cual se presenta un nuevo AE o una variación de uno AE existente, en primer lugar se presenta la nueva versión y sus modificaciones.

2. A continuación, se describen los algoritmos que se van a comparar. Si se trata del análisis de una nueva versión, lo más común es compararlo frente a la versión anterior. Si se presenta un nuevo algoritmo se eligen los que hasta ese momento hayan obtenido los mejores resultados o aquellos que sean más representativos.

3. Una vez que se conocen los algoritmos, se presenta el conjunto de funciones de prueba que se van a utilizar.

4. También es necesario elegir y describir el conjunto de medidas de rendimiento para comparar los AEs.

5. Antes de presentar los resultados obtenidos, se describen las condiciones en las que se han ejecutado las pruebas: número de ejecuciones, criterio de parada para los algoritmos, parámetros de los mismos, etc. Estos resultados se muestran en forma de tablas o gráficas.

6. Para terminar, se lleva a cabo la comparación y la explicación de las conclusiones que se extraen de los experimentos realizados. Generalmente, en este apartado se hace especial énfasis en los buenos resultados de la nueva propuesta.

Ya en 1995, Hooker [Hooker, 1995] criticaba la forma en la que se realizaban las comparaciones entre algoritmos metaheurísticos. El mecanismo "competitivo" que solía emplearse no resultaba útil desde el punto de vista de la investigación, ya que los resultados que se presentaban hacían énfasis en qué algoritmos obtenían los mejores resultados pero no en en el porqué de los mismos. Los más común era publicar únicamente los resultados de la competición y no los detalles que realmente contenían información importante. Este tipo de competiciones entre algoritmos provocaba, y continúa provocando, que se desviasen esfuerzos de investigación hacia el desarrollo de algoritmos cada vez más rápidos, desde el punto de vista únicamente computacional, o hacia encontrar el conjunto de parámetros que obtuviesen los mejores resultados, provocando que no se realizase un trabajo real de análisis e investigación de algoritmos.

Además de los problemas que Hooker mostró en su trabajo, el método habitualmente utilizado para analizar y comparar algoritmos presenta otros puntos críticos [Eiben and Jelasity, 2002]:

- En cuanto al conjunto de funciones de prueba que se utilizan generalmente se seleccionan "ad-hoc" para destacar los buenos resultados del algoritmo que se quiere presentar o analizar.
- Además, la categorización o caracterización de los conjuntos de funciones de prueba no siempre es correcta y normalmente se utilizan categorias binarias, por ejemplo funciones unimodales frente a multimodales, sin entrar en más detalle sobre las características de las funciones.
- Al igual que ocurre con la selección de funciones de prueba, la forma en la que se realizan las ejecuciones no siempre es justo y tiende a favorecer a unos algoritmos más que a otros.
- Aunque no ocurre siempre, es común que una vez realizadas las pruebas simplemente se muestren los resultados obtenidos sin relacionarlos con el funcionamiento de los algoritmos.
- A la hora de presentar conclusiones, estas se hacen de forma general, sin relacionar los resultados con el tipo de problema concreto o con el funcionamiento del algoritmo. Además, las conclusiones que se presentan suelen demostrar los buenos resultados de los algoritmos sin profundizar en los malos y las posibles soluciones para estos.

Con el objetivo de solventar estos problemas y de desarrollar una metodología estandarizada que permita obtener resultados de mayor utilidad cuando se analizan y comparan algoritmos evolutivos, en los últimos años se han presentado numerosos trabajos. El principal problema detectado tras la realización de una revisión exhaustiva de dichos trabajos es que los investigadores se han centrado en solucionar aspectos independientes del problema global de la caracterización formal. Estos aspectos se pueden agrupar en tres grandes temas: el diseño y estudio de funciones o problemas de prueba, el análisis estadístico de resultados y el diseño de experimentos [García et al., 2009]. En las siguientes secciones se analizará cada una de ellos, comenzando por el diseño y estudio de funciones de prueba.

Diseño y estudio de problemas de prueba (*Benchmarking*)

Los resultados y, ante todo, las conclusiones que se obtienen a partir de las pruebas que se realizan son muy dependientes del conjunto de funciones que se utilizan en los trabajos. Además, si se desea mejorar el rendimiento de los AEs existentes, es necesario desarrollar un conjunto de funciones de prueba que contenga funciones de la dificultad y las características deseadas.

Como ya se ha mencionado en secciones anteriores, el campo de aplicación de este trabajo es el de la optimización de funciones de codificación real. Dentro de este campo, en los primeros trabajos sobre análisis de AEs se utilizaba el conjunto de funciones de prueba que De Jong presentó en su tesis [De Jong, 1975], el cual incluía cinco funciones: una función unimodal, una función no lineal de dos variables, una función discontinua, una función continua y una función con óptimos locales (multimodal). A pesar de que a lo largo de los años se han añadido otras funciones [Ackley, 1987, Schaffer et al., 1989, Mühlenbein and Schlierkamp-Voosen, 1993], estos conjuntos no resultan útiles a la hora de realizar una comparación justa y útil entre algoritmos, debido, sobre todo, a que la mayoría de estas funciones son de tipo separable (este concepto será analizado en detalle más adelante). Es decir, no existen interacciones entre las variables de los problemas. Por lo tanto, el valor óptimo de cada variable se puede obtener de forma independiente al resto de las variables haciendo que el proceso de resolución de las mismas sea muy sencillo. Además, estos primeros conjuntos de funciones incluían funciones de baja dimensionalidad que también resultan sencillas para los AEs.

Por los motivos comentados anteriormente comenzaron a desarrollarse trabajos centrados en el diseño de nuevas funciones de prueba. Los primeros trabajos desarrollados en este sentido fueron presentados por Whitley [Whitley et al., 1996, Whitley et al., 1995]. El objetivo no consistía en desarrollar nuevos conjuntos de prueba, sino en proponer unos principios básicos para su desarrollo. En sus trabajos sugería que las funciones de prueba debían de ser representativas de problemas reales. También debían de ser funciones que presentasen dificultades para así conocer los límites de los algoritmos que se analizaban. Debían de ser conjuntos abiertos con distintos tipos de funciones, en el sentido de que permitiesen el planteamiento de hipótesis en cuanto al rendimiento de los AEs frente a estos grupos de funciones y la verificación de las mismas. Propusieron una guía para el desarrollo conjuntos de funciones, donde establecieron que un buen conjunto de funciones de prueba debía contener:

- Funciones que sean difíciles de resolver mediante métodos de búsqueda local.
- Funciones no lineales, no separables y no simétricas, es decir, con interacciones entre los parámetros que las definen.
- Funciones escalables en términos de dimensionalidad, es decir, en número de parámetros.
- Funciones escalables en cuanto a tiempo de evaluación.
- Funciones que sean configurables, pero donde el número de parámetros necesarios para definirlas sea bajo.
- Funciones que sea posible generarlas de forma aleatoria y que además la ingeniería inversa de las mismas sea difícil.
- Funciones cuya superficie de calidad posea características topológicas medibles.

Existen intentos por solucionar los problemas expuestos por Withley y Hooker y conseguir que las comparaciones y los análisis de AEs sean útiles para continuar con el desarrollo y la mejora de AEs existentes. El profesor P.N. Suganthan y otros investigadores del campo de la CE son los encargados de organizar cada año competiciones de AEs para la IEEE en las que las capacidades de los algoritmos se evalúan frente a diferentes tipos de funciones: funciones mono-objetivo de codificación real [Suganthan et al., 2005], funciones mono-objetivo de codificación real con restricciones [Liang et al., 2006, Mallipeddi and Suganthan, 2010], functiones multiobjetivo [Huang et al., 2007,Zhang et al., 2008], funciones de altas dimensiones (más de 100 variables) [Tang et al., 2007, Tang et al., 2009] o problemas de optimización con funciones dinámicas [Li et al., 2008]. En estas competiciones los conjuntos de problemas están limitados a cierto tipo de funciones tratando de analizar el comportamiento de los algoritmos y de sus estrategias de búsqueda en un tipo concreto de problemas, sin embargo, las clasificaciones que se realizan siguen siendo "grosso" modo y solamente en el caso de las funciones con restricciones se utiliza un valor numérico, el ratio entre la región factible y el espacio de búsqueda, que da una idea sobre el grado de dificultad de la función.

Otro de los objetivos del desarrollo de conjuntos de funciones de prueba es permitir el análisis de los algoritmos frente a funciones con características conocidas y medibles con el fin de entender cómo afectan dichas características concretas a las estrategias de búsqueda de los algoritmos. En [Eiben and Jelasity, 2002] se propone el uso de *clases de problemas*, que son conjuntos de funciones que comparten características que las diferencian de otros. Las clases de problemas deberán de mantener un equilibrio entre la generalidad y la concreción de las funciones que incluyan. Deberán de ser lo suficientemente generales como para sacar conclusiones razonables acerca del comportamiento de los algoritmos, pero también lo suficientemente concretas como para que sea posible distinguir el comportamiento de los algoritmos en cada clase de problema. Las instancias de las funciones de cada clase de problemas serán diferentes pero tendrán al menos alguna característica común. Por ejemplo, clasificar las funciones en problemas NP-completos o no NP-completos es irrelevante, es demasiado general como para poder obtener conclusiones útiles de la ejecución de un algoritmo. Lo mismo ocurriría si clasificásemos las funciones por su ámbito de aplicación, por ejemplo, problemas de clasificación. Los autores también critican que la clasificación típica que a día de hoy continua utilizándose y que organiza las funciones en unimodales / multimodales o separables / no separables por ser, de nuevo, demasiado general. Las conclusiones que se obtienen a partir de este trabajo es que todavía, a día de hoy, las funciones de prueba no están clasificadas correctamente.

Además de los intentos por clasificar las funciones de prueba en *clases de problemas*, en los últimos años han surgido numerosos trabajos que, en lugar de clasificar las funciones existentes, desarrollan herramientas que permiten la generación de problemas con ciertas características estructurales, las cuales es posible configurar con el fin de generar funciones con mayor o menor grado de dificultad y así estudiar el comportamiento de los algoritmos.

En este campo es bien conocido el trabajo de Kauffamn y los *NK-landscapes* (superficies de calidad NK) [Kauffman, 1993, Altenberg, 1997b]. Los *NK-landscapes* se definen como espacios de calidad generados aleatoriamente donde N es la dimensión del problema y K mide el grado de correlación entre los parámetros del mismo. Variando los valores de N y K se controla el grado de dificultad de los problemas. Inicialmente, los modelos *NK* fueron propuestos para generar superficies de calidad en problemas con codificación binaria, sin embargo ya existen trabajos donde se generan espacios de calidad con codificación real [Wang and Li, 2008]. También se han desarrollado extensiones del modelo original como [Altenberg, 1994, Altenberg, 1997b, Heckendorn and Whitley, 1997, Smith and Smith, 1995] donde,

además de utilizar los parámetros N y K, se añade un parámetro P que indica el número de particiones del espacio de búsqueda que contribuye a la calidad total.

En [Stuckman, 1988], Stuckman presentó un generador de problemas de optimización continua basado en combinaciones de funciones *sinc* cuadráticas que generaban superficies de calidad con muchos óptimos locales donde existían amplias zonas no derivables. Whitley *et al.* [Whitley et al., 1996, Whitley et al., 1995] desarrollaron un método basado en la composición de funciones conocidas que permitía obtener problemas no separables, no simétricos y escalables. En [Kennedy and Spears, 1998], los autores presentaron un generador de funciones multimodales para codificación binaria. Para generar la función se seleccionan P individuos del espacio de búsqueda que serán óptimos locales de la función y se les asignará el valor de calidad máximo o mínimo dependiendo de si el problema es de maximización o de minimización. Al resto de los individuos del espacio de búsqueda se le asigna un valor de calidad que depende de la distancia Hamming al óptimo local más cercano. Este modelo se extendió para generar funciones multimodales en espacios continuos [Kennedy, 1997]. También destaca el trabajo presentado en [Gallagher and Yuan, 2006], donde los autores desarrollaron un generador de funciones basado en funciones de tipo gaussiano que, utilizando pocos parámetros de configuración, permite obtener funciones con diferentes características topológicas.

También se han desarrollado generadores de funciones para problemas más específicos, como generadores para problemas dinámicos [Morrison and De Jong, 1999], problemas de optimización con restricciones [Michalewicz et al., 2000], donde se permite la parametrización del ratio entre la región factible y el espacio de búsqueda total, la conexión de las regiones factibles y el número de restricciones; o un generador de problemas multiobjetivo [Weise et al., 2008] donde, además de modificar el número de objetivos, también se permite variar el grado de interacción entre los parámetros de las funciones (igual que en los modelos *NK*), el nivel de neutralidad de la función (zonas donde no existe variación en el valor de calidad) y la rugosidad (número de óptimos locales de la función).

A partir de esta revisión sobre los trabajos realizados en cuanto a diseño y estudio de problemas de pruebas se obtienen dos conclusiones principales:

- En primer lugar, existen grandes esfuerzos a la hora de tratar de clasificar las funciones de prueba existentes en términos que resulten realmente útiles para analizar el comportamiento de los algoritmos.

- En segundo lugar, el intento de desarrollar generadores de funciones de prueba configurables es también un campo de interés para los investigadores.

Análisis estadístico de resultados

Como dicen los autores en [Moreno-Pérez et al., 2007] el análisis y comparación de algoritmos metaheurísticos (en los que se incluyen los AEs) deberá realizarse teniendo en cuenta qué propiedades se buscan para esos algoritmos. No es posible mejorar todas las propiedades al mismo tiempo, ya que muchas están relacionadas. Puede ocurrir que la mejora de una provoque que otra se vea perjudicada. En cuanto a las propiedades que los investigadores pueden desear para sus algoritmos, en el mismo trabajo se clasifican en los siguientes términos:

- En términos de facilidad de comprensión y aplicación, se tienen las características de simplicidad, precisión y coherencia.

- Si nos basamos en su aplicabilidad, las características deseadas serían generalidad, adaptabilidad y robustez.

- Si buscamos un algoritmo que pueda ser utilizado de forma favorable en un sistema de ayuda a la toma de decisiones, buscaríamos un algoritmo que cumpliese las propiedades de interactividad, multiplicidad y autonomía.

- Si basamos nuestras decisiones en el rendimiento práctico, analizaríamos la eficiencia, la efectividad y la eficacia.

De todas estas características, las que son fácilmente cuantificables y son objeto de análisis en la mayor parte de los trabajos presentados son: eficiencia, efectividad, eficacia y robustez. La eficiencia se mide en términos de la cantidad de recursos empleados, tanto de espacio como, sobre todo, de tiempo de cómputo. La eficacia se define como la probabilidad de alcanzar una solución óptima. La efectividad se mide como la calidad de las soluciones propuestas por el algoritmo. Por último, la robustez se mide como la variabilidad del comportamiento de un algoritmo dado, es decir, cómo de estable es a la hora de resolver un problema.

Debido a la naturaleza estocástica de los AEs, para el análisis de los resultados que obtienen tras las ejecuciones son necesarias medidas estadísticas. Qué medidas utilizar para cuantificar el rendimiento y analizar estos resultados depende, en menor o mayor medida, del objetivo para el cual se va a utilizar el algoritmo evolutivo y con esta premisa los investigadores deberían escoger las medidas más adecuadas. Algunas de las más utilizadas son las siguientes [Eiben and Jelasity, 2002]:

- **Tasa de éxito o Success Rate (SR):** se utiliza cuando se conoce la solución óptima del problema y se define como el porcentaje de ejecuciones que terminan con éxito. Sirve para analizar los algoritmos en términos de robustez y eficacia.

- **Mejor calidad media o Mean Best Fitness (MBF):** en la ejecución de un AE, el mejor valor de calidad se define como el valor de calidad del mejor individuo cuando finaliza la ejecución. Este valor representa la media de la calidad de los mejores individuos en cada una de las ejecuciones que se realicen. Esta medida se utiliza para analizar los algoritmos en función de su efectividad.

- **Número medio de evaluaciones o Average number of Evaluations to a Solution (AES):** se utiliza para comparar y analizar los algoritmos en términos de eficiencia: mide el número de evaluaciones que son necesarias hasta obtener un determinado nivel de calidad. De esta forma, podemos estimar qué algoritmo tiene una estrategia de búsqueda más rápida sin tener el cuenta el tiempo que cuesta la evaluación de las soluciones, haciendo que el análisis de la eficiencia de un algoritmo sea independiente de implementaciones hardware y de recursos software, centrándonos únicamente en sus capacidades.

Si nos fijamos en la mayor parte de los trabajos presentados, los análisis se realizan basándose en resultados medios sobre un número prefijado de ejecuciones. Por ejemplo, en el trabajo donde se presenta el DE [Storn and Price, 1997] en el apartado de resultados, los autores escogen tres técnicas y tres conjuntos diferentes de funciones de prueba para demostrar los buenos resultados de su algoritmo, pero en ningún caso se obtienen conclusiones más allá de las comparaciones numéricas ni se justifica la elección de las tres técnicas ni de

los conjuntos de prueba. No existen muchos trabajos que utilicen procedimientos estadísticos para comparar resultados, aunque en los últimos años su uso está creciendo debido a las recomendaciones de algunos revisores y expertos. De entre aquellos trabajos que sí realizan procedimientos estadísticos para analizar sus resultados los hay que utilizan test paramétricos y otros que utilizan test no paramétricos.

Destacan tres trabajos que utilizan test paramétricos en sus análisis [Czarn et al., 2004,Ozcelik and Erzurumlu, 2006, Rojas et al., 2002]. En el primero de ellos, los autores proponen una metodología estadística para el estudio de AGs y analizan como afectan los parámetros a los resultados, para ello utilizan un análisis de varianza (ANOVA). Además, utilizan regresión polinómica para obtener la curva de respuesta de los parámetros, si hay interacción entre los parámetros una curva total y si no existe interacción una curva individual. En [Rojas et al., 2002] también analizan el comportamiento de AGs y la importancia de sus parámetros en cuanto a su influencia en el balance exploración / explotación. Estudian también las interacciones y las interrelaciones entre los parámetros. A pesar de este análisis profundo, determinar los parámetros para un problema concreto es todavía una línea de investigación abierta. Por último, los autores de [Ozcelik and Erzurumlu, 2006] aplican una solución basada en redes de neuronas artificiales y algoritmos genéticos al problema de la minimización de deformaciones en piezas de plástico. Para analizar los resultados obtenidos y obtener conclusiones estadísticamente significativas utilizan, de nuevo, ANOVA.

Los test no paramétricos se utilizan cuando los experimentos con AEs no verifican las hipótesis de partida necesarias para el uso de test paramétricos. En [García et al., 2007] se utilizaron los test de Friedman, Iman-Davenport, Bonferroni-Dunn y Wilcoxon para analizar los resultados de un AG. Los autores de [Moreno-Pérez et al., 2007] compararon los resultados de varias metaheurísticas utilizando el test de rangos con signos de Wilcoxon y el test de Friedman. Para después utilizar Namenyo y Bonferroni-Dunn y obtener conclusiones sobre dichas comparaciones con los procedimientos de Holm y Hochberg. Los resultados presentados en la competición del *CEC'2005* [Suganthan et al., 2005], fueron analizados en [García et al., 2009] utilizando test no paramétricos para comparar los resultados de la resolución de un solo problema o de varios problemas. Los test que se utilizaron fueron, de nuevo, el test de Friedman, Iman-Davenport, Bonferroni-Dunn, Holm, Holchbert y Wilcoxon.

De los trabajos presentados en este área se obtienen las siguientes conclusiones:

- Antes de llevar a cabo el análisis de un AE es necesario establecer los objetivos de dicho análisis para elegir de forma adecuada las medidas que se van a utilizar.

- Los análisis de resultados medios no son suficientes para obtener conclusiones correctas acerca de las pruebas realizadas. Debe de generalizarse el uso de análisis estadísticos basados en contrastes de hipótesis.

Diseño de experimentos

Los AEs no sólo son técnicas de optimización configurables, sino que además los valores óptimos de los parámetros de configuración son altamente dependientes de la clase de problemas que se vayan a resolver. Existen numerosos trabajos que tratan de reducir el número de parámetros en los AEs e incluso desarrollos sobre AEs auto-configurables. En [Eiben et al., 2007] se hace una revisión sobre posibles maneras de configurar los algoritmos. Hasta que llegue el momento en el que no sea necesario ningún parámetro de configuración, los investigadores tienen que tratar con el problema de configurar sus algoritmos de la mejor manera

posible para obtener los mejores resultados. En este sentido, el *Diseño de experimentos* (DoE) es una técnica de estadística que permite analizar la influencia de los parámetros en los algoritmos para realizar un ajuste óptimo de los mismos de acuerdo con el problema para el cual será aplicado. El diseño de experimentos surgió a principios del siglo XX y fue aplicado por Ronald Fisher [Fisher, 1966] en sus estudios de agronomía. Muchas de las aplicaciones en las que se ha utilizado son en el campo de la biología pero, debido a su éxito, y, sobre todo, a su formalismo su uso ha ido creciendo en otros campos.

En el campo de la CE su uso es reciente y no existen muchos trabajos en los cuales se aplique el diseño de experimentos para analizar los algoritmos presentados y la influencia de los parámetros de configuración. Sin embargo, en los últimos años han surgido varias metodologías basadas en los principios de DoE con el objetivo de proporcionar marcos de análisis y comparación de estándares de forma que las conclusiones que se extraigan de los trabajos sean útiles y, ante todo, sean de provecho para desarrollos futuros.

Los primeros trabajos que trataron de utilizar una metodología estándar, además de basarse en las ideas de DoE, utilizaban también las metodologías que se aplicaban en Aprendizaje Máquina. En esta disciplina, para analizar el comportamiento de un algoritmo frente a una clase de problemas y encontrar los parámetros que permitían obtener los mejores resultados, se seguían los pasos que se describen a continuación:

1. Se generaba el conjunto de entrenamiento a partir de instancias de la clase de problema elegida.

2. Se generaba el conjunto de prueba con instancias de la clase de problema que no estaban en el conjunto de entrenamiento.

3. De forma iterativa, se ejecuta el algoritmo frente al conjunto de entrenamiento y se obtienen los parámetros que proporcionan los mejores resultados.

4. Para validar estos parámetros, se ejecuta el algoritmo frente al conjunto de test y se comprueba si de verdad obtiene los mejores resultados.

El punto clave de esta técnica es que la validación de los resultados se realiza sobre instancias de la clase de problema que no están en el conjunto de entrenamiento y que no han sido vistas anteriormente por el algoritmo.

Entre las metodologías más recientes destaca *F-RACE* [Birattari et al., 2002] que proporciona un procedimiento con el cual se puede obtener la configuración óptima de una metaheurística dependiendo de la clase de problemas que se quiera resolver. El procedimiento presentado es un mecanismo experimental que se guía por medidas estadísticas. Su implementación se basa en eliminar, en las primeras etapas del análisis, las configuraciones que no proporcionen buenos resultados, reduciendo drásticamente el número de candidatos para, de esta forma, acelerar el proceso de búsqueda. Las principales aportaciones de este método son, en primer lugar, una definición formal del problema de configuración de una metaheurística y, en segundo lugar, la verificación de que es posible la configuración de forma eficiente y efectiva en un proceso rápido.

REVAC [Nannen and Eiben, 2007] es otra de las metodologías basada en diseño de experimentos que se ha desarrollado en los últimos años. En la mayor parte de los trabajos, el ajuste de los parámetros de un AE se basa en convenciones que no están correctamente justificadas y en métodos "ad-hoc". Los investigadores suelen hacer un barrido en los rangos de los parámetros del algoritmo, tan amplio como sus capacidades computacionales les permita.

REVAC se basa en la teoría de la información para medir la relevancia de los parámetros estimando el rendimiento esperado para un conjunto de parámetros tratando de maximizar la entropía de Shannon. La maximización de esta entropía se utiliza como medida de la relevancia de los parámetros. El método se basa en medir cuánta información es necesaria para alcanzar un cierto nivel de rendimiento y cómo la cantidad de información depende de cada uno de los parámetros del algoritmo. Desde un punto de vista más técnico, la metodología REVAC utiliza un algoritmo de tipo EDA diseñado para maximizar la entropía en dominios continuos. De forma iterativa se ajusta la distribución de probabilidad de cada uno de los parámetros a configurar y de esta forma estimar la relevancia de cada uno de ellos. Uno de los últimos trabajos desarrollados con la metodología REVAC [Smit and Eiben, 2010] consigue mejorar los resultados obtenidos por el algoritmo G-CMA [Auger and Hansen, 2005] en la competición celebrada en el congreso *CEC'2005* [Suganthan et al., 2005].

SPO [Bartz-Beielstein and Preuss, 2007] es una metodología que permite, mediante experimentación, configurar algoritmos y obtener conclusiones acerca de dicha configuración. Utiliza técnicas estadísticas para obtener resultados y conclusiones fiables. El ciclo de trabajo de SPO sigue los pasos que se describen a continuación:

1. La investigación inicial parte de hipótesis acerca de la influencia que pueden tener ciertos parámetros en el comportamiento de los algoritmos.

2. Esas suposiciones se formulan como hipótesis estadísticas.

3. Para cada hipótesis se realizan los experimentos necesarios. Se elige el modelo y el diseño experimental que se van a utilizar. Se generan los datos y se ajusta el modelo hasta que las hipótesis puedan ser rechazadas o aceptadas.

4. Se obtienen las conclusiones a partir de los resultados de los experimentos y de las hipótesis aceptadas o rechazadas.

Para el tercer paso de esta metodología, la ejecución de los experimentos, se ha implementado una herramienta software SPOT (*Sequential Parameter Optimization Toolbox*) [Bartz-Beielstein and Preuss, 2010, Bartz-Beielstein, 2010]. Esta herramienta ha sido aplicada con éxito en campos como la bioinformática [Fober et al., 2009], la industria aeroespacial [Naujoks et al., 2006], el análisis estadístico de algoritmos [Trautmann and Mehnen, 2009] o la optimización multimodal [Preuss et al., 2007].

Esta tercera técnica es, a día de hoy, la menos aplicada en el campo de los AEs. Sin embargo, es obvia su necesidad, debido a que el funcionamiento de los AEs es muy dependiente de su configuración.

Como conclusión a este apartado dedicado a mostrar la falta de procedimientos estándar de caracterización formal de AEs y sus consecuencias a nivel práctico, debemos decir que, si bien existen investigadores que están centrado sus esfuerzos en atacar este problema, realmente se ha avanzado poco en el desarrollo de un procedimiento general. Así, los trabajos se pueden englobar en tres grandes temáticas que han sido estudiadas y probadas pero de forma independiente, sin que hayamos encontrado intentos destacables de solventar el problema de la caracterización formal en su totalidad.

3.3 Conclusiones

A lo largo de este apartado se ha desarrollado el marco teórico en el que se engloba la presente tesis doctoral. Se ha tratado de dejar patente cómo en los últimos 20 años han aumentado de forma muy significativa los campos de aplicación de los AEs tratando de solucionar problemas cada vez más reales y, por tanto, más complejos. Esto ha motivado el desarrollo constante de mejoras en los algoritmos existentes y la aparición de otros nuevos. Pero todos estos años de innovación no han venido acompañados de estudios teóricos sobre las características de los problemas y las soluciones aportadas en los algoritmos de una forma estandarizada y formal que permita a los usuarios de otros campos tales como la Ingeniería, la Física o la Biología conocer a priori si un cierto algoritmo evolutivo es adecuado para sus problemas o no. Esto conlleva un proceso de prueba y error que sería evitable si los desarrolladores de nuevos algoritmos los caracterizasen formalmente antes de presentarlos a la comunidad científica.

En los siguientes apartados de esta tesis se afronta este problema, en una primera aproximación, proponiendo un procedimiento de caracterización que permita que los desarrolladores presenten sus AEs en función de ciertas propiedades de los espacios de calidad, que serán seleccionadas por su relevancia, para problemas de optimización con parámetros reales. El procedimiento de caracterización que se presenta se basa en las conclusiones extraídas del análisis de la bibliografía relativa a la caracterización formal de los algoritmos evolutivos y comentada en la sección anterior. Así, el procedimiento propuesto consta de tres etapas coincidentes con los tres grandes tipos de problemas detectados: selección y caracterización de las funciones de prueba, establecimiento de medidas de análisis representativas y creación de una metodología de análisis de resultados que sea general y proporcione conclusiones útiles a nivel práctico.

Capítulo 4

Procedimiento de caracterización

El principal objetivo de este trabajo es el planteamiento de un procedimiento de caracterización de algoritmos evolutivos que permita que los diseñadores de los mismos analicen sus desarrollos y presenten los resultados de forma homogénea y formal. En el capítulo anterior, tras la revisión bibliográfica de los trabajos con un objetivo similar en este campo, se detectaron los tres problemas clave en un proceso de caracterización de algoritmos: la correcta elección del conjunto de funciones prueba, la determinación de las medidas de análisis representativas y el establecimiento de una metodología ejecución de las pruebas y de análisis de los resultados. El procedimiento de caracterización que aquí se plantea se basa en tres etapas que deben afrontar y solucionar estos tres problemas, pero no de forma independiente, sino creando un flujo de trabajo con dependencias entre unas y otras. La figura 4.1 muestra un esquema de las etapas que constituyen el procedimiento y su interrelación:

1. Caracterización de espacios de calidad: en esta etapa se debe seleccionar el conjunto de funciones de prueba (benchmark) de forma que sea representativo del tipo de problema a resolver, en el marco de este trabajo, optimización de funciones con codificación real. Para ello, es necesario establecer las características de los espacios de calidad más relevantes y analizar en base a ellas las funciones seleccionadas.

2. Medidas de error y rendimiento: se deben seleccionar las medidas sobre las que se analizarán los resultados de los algoritmos evolutivos al ser ejecutados en las funciones seleccionadas en la etapa anterior. Este problema puede ser tratado de forma concurrente con el primero.

3. Estudio experimental: los resultados de las ejecuciones de los algoritmos sobre las funciones de prueba deben ser analizados en base a las características establecidas en la primera etapa y utilizando las medidas seleccionadas en la segunda. Por ello, esta etapa debe ser llevada a cabo tras la finalización de las dos anteriores. En este trabajo se planteará una metodología de ejecución de las pruebas y análisis de los resultados que componen la parte experimental del procedimiento.

A lo largo de este capítulo se propondrán soluciones concretas en cada una de estas tres etapas. Dichas soluciones son una primera aproximación a la resolución global del problema de caracterización formal de algoritmos evolutivos, y deberán ser extendidas en el futuro. De cualquier forma, suponen un primer intento de solución general de este problema y, como se demostrará en la sección de Aplicación, proporcionan resultados muy satisfactorios.

En la primera sección de este capítulo, dedicada a la caracterización de espacios de calidad, se describirá dicho concepto y sus principales características para concluir que, en el marco de este trabajo, las características a estudiar de los espacios de calidad son la separabilidad y la modalidad. A continuación se presentará con detalle el conjunto de funciones de prueba que se utilizará en la sección de aplicación. Finalmente, se explicarán los algoritmos desarrollados en este trabajo a la hora de caracterizar los espacios de calidad en términos de separabilidad y modalidad. Para continuar, en la segunda sección del capítulo, se proponen las medidas de error y rendimiento a utilizar para analizar los resultados de las ejecuciones de forma homogénea. En la tercera sección de este capítulo se describirán los elementos que constituyen el estudio experimental. Primeramente, se deben establecer unos parámetros de ejecución objetivos y, a continuación, se debe plantear una metodología de análisis formal de resultados. En este sentido, se proponen tres pasos consecutivos: análisis con la configuración básica, análisis con modificaciones de parámetros y análisis comparativo.

Antes de comenzar con el desarrollo de cada una de las etapas, debemos enfatizar que en estas se utilizarán algunas de las aportaciones de otros investigadores comentadas en la sección anterior a nivel de diseño y estudio de funciones de prueba y de análisis estadístico de resultados. En cuanto al diseño de experimentos, este está centrado en la etapa de desarrollo de un algoritmo evolutivo, durante la cual el diseñador debe estudiar el comportamiento del mismo en función de sus parámetros de configuración. Por tanto, queda fuera del alcance de este procedimiento de caracterización, que ha sido desarrollado para una etapa posterior a la experimentación, la que se debe realizar una vez comprobado el funcionamiento del algoritmo de cara a proporcionar resultados formales y objetivos. En consecuencia, en la tercera etapa del procedimiento de caracterización, no se utilizarán técnicas de diseño de experimentos, sino que se propondrá una metodología totalmente nueva.

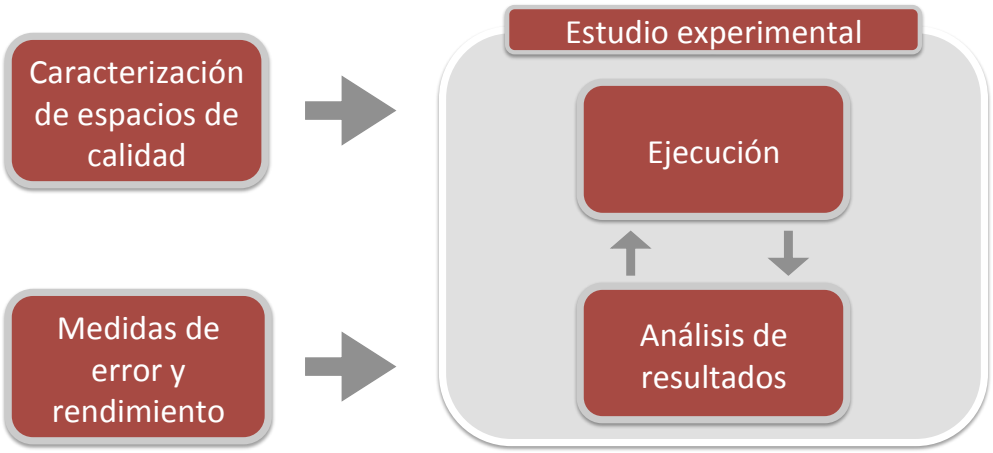


Figura 4.1: Elementos del procedimiento de caracterización formal presentado en este trabajo.

4.1 Caracterización de espacios de calidad

La primera de las etapas se centra en la correcta elección del conjunto de funciones prueba y su caracterización. Tras la revisión realizada en el capítulo 3 se llegó a la conclusión de que, para una correcta caracterización de un AE, además de utilizar un conjunto de funciones de prueba completo que incluya funciones con diferentes características y que, además, supongan un reto para los algoritmos, las propiedades de dichas funciones deberán ser analizadas

en profundidad. Por este motivo, la solución propuesta en este trabajo para esta primera etapa implica la caracterización formal de los espacios de calidad que generan dichas funciones de prueba. La elección y el ajuste de parámetros de los operadores de los AEs son clave para el éxito de los procesos de optimización y esta correcta configuración es altamente dependiente del espacio de calidad sobre el que el AE actuará. Por este motivo, proponemos una serie de algoritmos que permiten la caracterización de los espacios de calidad como paso previo a la aplicación de un AE. Para la explicación de los mismos es necesario, en primer lugar, definir el concepto de espacio de calidad y las características topológicas que tienen mayor influencia en el comportamiento de los AEs. Previo a la explicación de los algoritmos de caracterización desarrollados, se realizará una breve revisión de las medidas de caracterización de espacios de calidad utilizadas en la bibliografía para el análisis de los mismos.

4.1.1 Espacio de calidad (Fitness Landscape)

El concepto de espacio de calidad (del inglés, *fitness landscape*) fue introducido originalmente por S. Wright [Wright, 1932] como medio para visualizar las relaciones entre el genotipo de un individuo y su éxito o calidad reproductiva. Relacionado con esta idea de espacio está, por una parte, la función de calidad, que asigna a cada genotipo o individuo de una población un valor de calidad o bondad; y, por otro lado, el espacio de búsqueda o la disposición de los genotipos en un espacio abstracto, que proporciona una relación de vecindad entre dichos genotipos y, por consiguiente, permite estimar la facilidad o frecuencia con la que es posible trasladarse de un punto a otro de este espacio. Este concepto ha sido utilizado en otros campos, no relacionados con la CE como en la física de sistemas complejos, para estudiar las configuraciones de los vidrios de spin [Binder and Young, 1986, Mezard et al., 1987], en biofísica para estudiar los espacios de energía que gobiernan el plegado de biopolímeros [Chan and Dill, 1991, Dill et al., 1995] o los llamados espacios electorales (*Electoral landscapes*) que se utilizan para estudiar el comportamiento de los modelos espaciales de voto. [Kollman et al., 1997, Stadler, 1998]. En el dominio de la optimización, el interés principal de esta "metáfora" es el de relacionar la descripción topológica del espacio de calidad con la dinámica de los algoritmos de búsqueda y, de esta forma, extraer información sobre la dificultad de optimizar un problema dado.

Formalmente, un espacio de calidad [Radcliffe and Surry, 1995, Stadler, 2002, Stadler, 1995] se representa por una terna o tripleta $(\mathcal{S}, \mathcal{V}, f)$, tal que:

- $\mathcal{S}$, es el espacio de búsqueda o conjunto de soluciones potenciales a un problema dado.
- $\mathcal{V} : \mathcal{S} \rightarrow 2^{\mathcal{S}}$, es una función de vecindad que a cada solución $s \in \mathcal{S}$ le asigna un conjunto de vecinos $\mathcal{V}(s) \subset \mathcal{S}$.
- $f : \mathcal{S} \rightarrow \mathbb{R}$, es una función de aptitud o calidad que a cada solución le asigna un valor real que representa su grado de aptitud para resolver un problema dado.

Los espacios de calidad permiten representar el conjunto de soluciones a un problema, es decir, el espacio de búsqueda, y la bondad o calidad de cada una de dichas soluciones a la hora de resolver un problema concreto. En este caso, los "organismos potenciales" que Wright utilizaba en su definición para representar a los elementos del sistema, se corresponden con las posibles soluciones a un problema dado. El concepto de vecindad se relaciona con los operadores de búsqueda que utilice el algoritmo: dos organismos o soluciones serán vecinas si, aplicando dichos operadores a la primera solución, somos capaces de alcanzar la segunda.

Finalmente, el valor de calidad se corresponde con el valor objetivo que deseemos minimizar o maximizar. En la figura 4.2 muestra la relación entre un espacio de búsqueda y un espacio de calidad a través de una función de aptitud. El mismo espacio de búsqueda genera diferentes superficies de calidad, con distintas topologías, dependiendo de la función de aptitud del problema.

En el caso de las metaheurísticas, en las cuales se incluyen los AEs, el concepto de vecindario está relacionado con uno o varios operadores de búsqueda. Un vecindario es el conjunto de soluciones alcanzables después de aplicar los operadores del algoritmo. Matemáticamente: $\mathscr{V}(s) = \{y \in \mathscr{S} | y = op(s)\}$. De una forma abstracta, las metaheurísticas tratan las superficies de calidad como un sustrato por el cual se desplazan las soluciones. El concepto de espacio de calidad permite estudiar la dinámica de la evolución de soluciones, la convergencia de las metaheurísticas y la capacidad de los algoritmos a la hora de resolver un problema.

El concepto de los espacios de calidad sirve como medio para caracterizar la topología de un problema y, a continuación, identificar el algoritmo que saca el máximo partido de las características propias de las superficies que definen. Una descripción estadística de tales superficies sería capaz de discriminar decisiones equivocadas, donde pocos algoritmos pueden optimizar el problema correctamente o seleccionar los mejores parámetros posibles (operadores, codificación, etc.) que sean favorables para una optimización eficiente. Algunas herramientas matemáticas que permiten esta caracterización son el estudio de grafos que se obtienen a partir de este espacio de calidad [Reidys and Stadler, 2001] o la descomposición en series de Walsh [Kallel et al., 2001], ambas aplicadas a la caracterización de superficies de calidad con espacios de búsqueda discretos donde, además, es necesario conocer la expresión analítica de la función de calidad. Por lo tanto, estas herramientas no serían útiles para analizar problemas de aplicaciones reales donde, en muchas ocasiones, no se cuenta con dicha expresión analítica. En consecuencia, es necesario el desarrollo de herramientas que permitan realizar esta estimación estadística de propiedades sin la necesidad de conocer dicha expresión.

Para realizar una estimación correcta de la topología de un espacio de calidad, en primer lugar, es necesario seleccionar qué propiedades de dicho espacio son las que los definen o describen. Para tomar esta decisión, en el siguiente apartado se realizará una revisión de las características de las superficies de calidad más utilizadas cuando se trabaja con AEs. Una vez analizadas estas características, concluirá cuáles utilizar y cuáles no y en una sección posterior se presentarán los algoritmos de caracterización desarrollados para dicho fin.

Características topológicas de los espacios de calidad

Dentro de las características topológicas de los espacios de calidad, y tras revisar los principales trabajos del campo, se puede concluir que las más influyentes en el comportamiento de los AEs son: la modalidad, la rugosidad, la epístasis y la deceptividad. Algunas de estas características no aparecen de forma aislada y sus relaciones pueden aumentar la dificultad de un problema. En esta sección se hará una descripción de cada una de ellas centrándonos en su relevancia para el caso concreto de los AEs.

Modalidad La mayor o menor dificultad de un espacio de calidad está ligada, entre otros factores, a la existencia o no de óptimos locales y a la distribución de los mismos. Basándonos en la modalidad, las funciones que definen el espacio de calidad pueden dividirse en unimodales y multimodales. Una función o espacio de calidad es unimodal cuando no existe

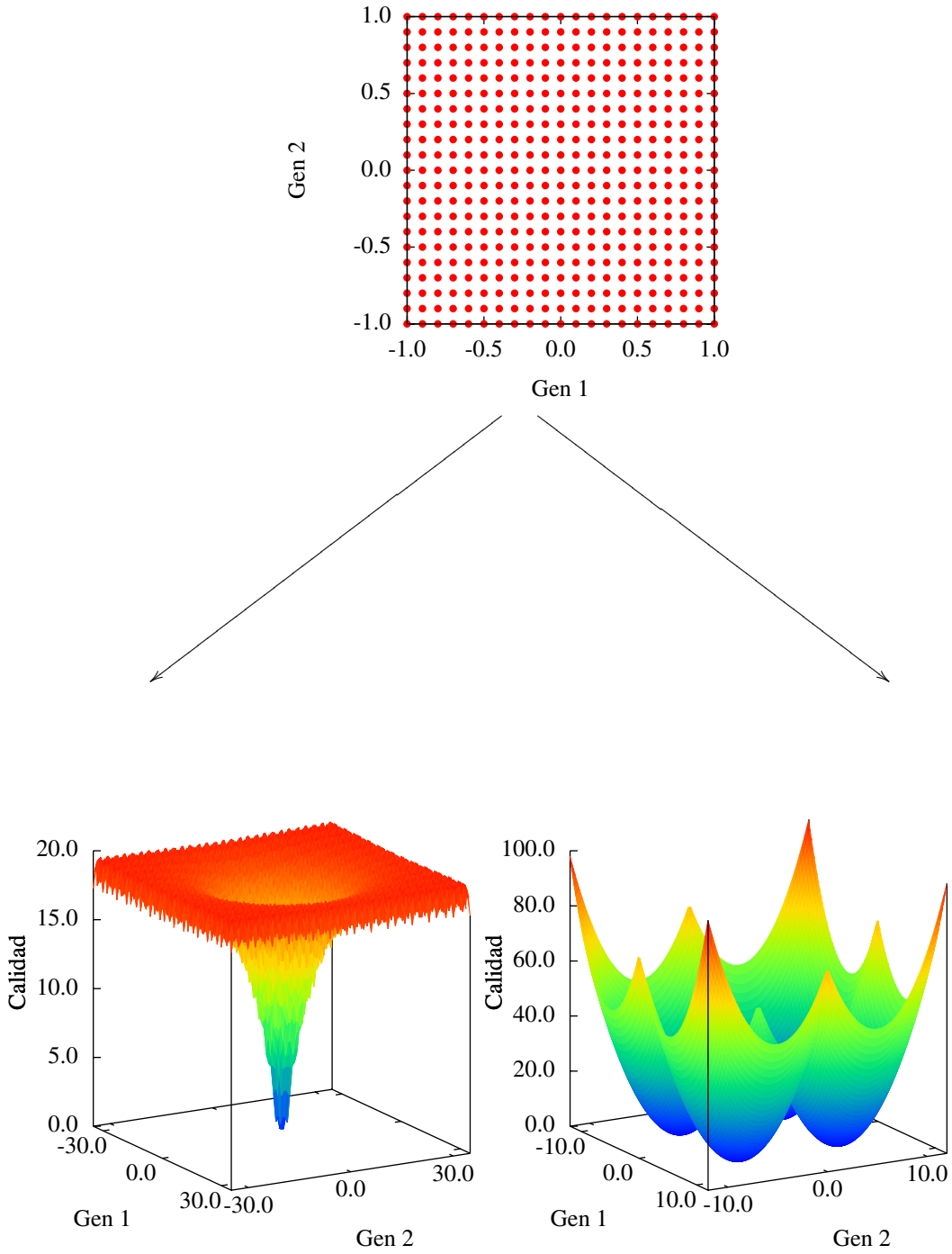
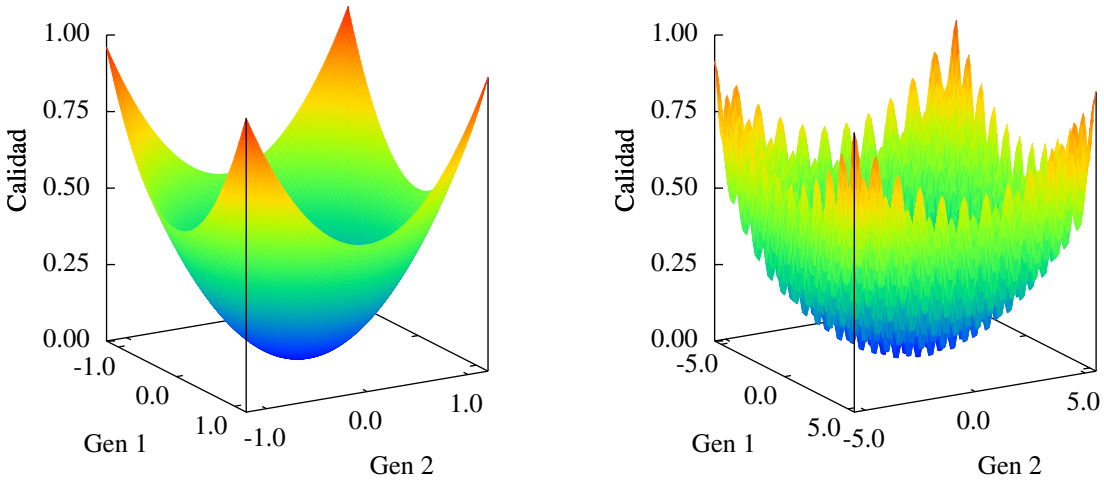


Figura 4.2: Representación de un espacio de búsqueda y de los posibles espacios de calidad asociados dependiendo de la función de calidad utilizada. Aunque en la imagen el espacio de búsqueda se muestre discretizado, este se corresponde con el espacio $\mathbb{R}^2$. Los espacios de calidad representados se corresponden con la función *Ackley's*, a la izquierda, y la función *Becker*, a la derecha.



(a) Espacio de calidad unimodal.

(b) Espacio de calidad multimodal.

Figura 4.3: Representación de dos espacios de calidad, uno unimodal y otro multimodal. La modalidad es una de las pocas características de los espacios de calidad que puede estudiarse gráficamente en funciones de hasta dos dimensiones de forma aproximada.

más que un óptimo global. Si por el contrario existe más de un óptimo global o, además del óptimo global, existen óptimos locales entonces la función se define como multimodal. En la figura 4.3 se muestran dos superficies de calidad: una unimodal y otra multimodal.

En el campo de la optimización se definen los conceptos de óptimo local y óptimo global de la siguiente forma:

- Una solución s^* del espacio de búsqueda es un óptimo local, también llamado óptimo relativo, si es una solución óptima en una zona contigua del espacio de calidad pero no es necesariamente la mejor solución del espacio de calidad completo. Es decir, en un problema de minimización, dado un entorno reducido y contiguo del espacio de calidad al cual pertenece s^*, $E(s^*)$, se define óptimo local como:

$$s^* \in \mathbb{R}^n \text{es un óptimo local} \Leftrightarrow \forall s_i \in E(s^*), f(s^*) \leqslant f(s_i) \tag{4.1}$$

- Una solución s^* del espacio de búsqueda es un óptimo global si, además de ser un óptimo local, es la que mejor valor de calidad presenta de entre todas las posibles soluciones. En un problema de minimización, matemáticamente se define de la siguiente forma:

$$s^* \in \mathbb{R}^n \text{es un óptimo global} \Leftrightarrow \forall s_i \in \mathbb{R}^n, f(s^*) \leqslant f(s_i) \tag{4.2}$$

Aunque los espacios de calidad que presentan varios óptimos locales suelen ser de mayor interés debido a su dificultad, en el caso de las funciones unimodales (aquellas que sólo presentan un óptimo) también existen casos con un alto grado de complejidad. Por ejemplo, cuando el óptimo está aislado y no hay información en el espacio de calidad, es difícil realizar una exploración satisfactoria de la misma que permita encontrarlo. En estos casos, no

existe ningún algoritmo de búsqueda que tenga un rendimiento mejor al de una búsqueda aleatoria [Horn and Goldberg, 1994]. Estos problemas también reciben el nombre de *needle-in-a-haystack* (NIaH), es decir, *aguja en un pajar* [Goldberg, 1989b]. En [Jones, 1995] los autores describen matemáticamente los problemas NIaH para codificación binaria como una función cuyo valor es siempre 0 excepto en un punto del espacio de búsqueda donde el valor que toma la función es 1. Este concepto se puede generalizar considerando que un problema NIaH es un problema cuya función de calidad toma siempre el mismo valor $x \pm \varepsilon \in \mathbb{R}$ (siendo ε un margen de error ya que se trabaja con números reales) en todos los puntos del espacio de búsqueda excepto en el punto óptimo donde toma otro valor $y < x$, en el supuesto de que estemos minimizando.

Como ya hemos mencionado antes, un espacio de calidad es multimodal si presenta más de un óptimo (M será el número de óptimos del espacio). En [Garnier and Kallel, 2000], Garnier y Kallel introducen el concepto de centro de atracción como elemento clave en la caracterización de la multimodalidad de los espacios de calidad. Los autores establecen que la complejidad de un espacio de calidad para un AE u otra metaheurística no reside únicamente en la existencia de más o menos óptimos locales, sino en su distribución y en la topología de las áreas entre ellos. Así, definen el concepto de centro de atracción como la zona del espacio que ``rodea" a un óptimo, de tal forma que una vez en dicho centro, cualquier descenso de gradiente lleva siempre al óptimo asociado. La relevancia de este trabajo se encuentra en el planteamiento del espacio de calidad como una zona particionada con transiciones más o menos abruptas entre óptimos, y no como una simple colección de óptimos aislados. Resulta evidente que la estimación de los centros de atracción será siempre aproximada, más cuanto mayor sea la dimensionalidad del espacio, pero los autores demuestran que es un concepto muy importante para analizar la complejidad de un espacio multimodal. Por este motivo, será utilizado como elemento clave en este trabajo

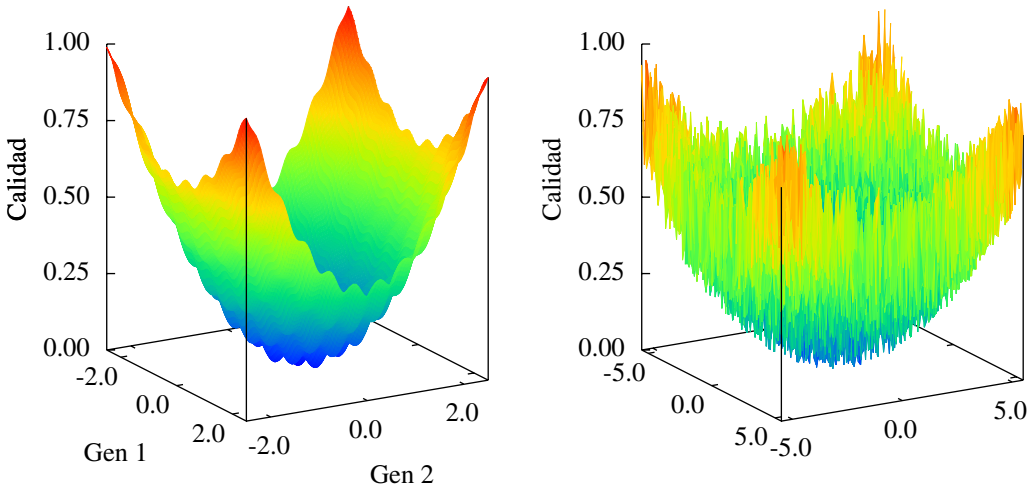
Rugosidad La rugosidad de un espacio de calidad está directamente relacionada con su estructura más o menos "accidentada" y es un factor que también influye en la menor o mayor dificultad para solucionar un problema. Si un espacio es irregular, entonces se dice rugoso; en el caso contrario, se dice que el espacio es continuo o liso. En la figura 4.4 se muestran dos ejemplos de espacios de calidad con diferentes grados de rugosidad: en la figura de la izquierda se muestra un espacio de calidad con poca rugosidad, casi liso, y en la figura de la derecha se muestra un espacio de calidad con un alto nivel de rugosidad.

La cuantificación de la rugosidad de un espacio se realiza midiendo la correlación entre los valores de calidad de las soluciones vecinas. Las funciones de *autocorrelación* y la *longitud de correlación* [Weinberger, 1990, Weinberger, 1991] son los indicadores de rugosidad más accesibles para cálculo numérico. La función de autocorrelación $\rho(n)$ de la "serie temporal" de valores de calidad ($f(x_t)$, $f(x_{t+n})$, …) de una serie de puntos aleatorios $x_t, x_{t+n}, \ldots$ se define como:

$$\rho(n) = \frac{E[f(s_t)f(s_{t+n})] - E[f(s_t)]E[f(s_{t+n})]}{\mathrm{var}(f(s_t))} \tag{4.3}$$

donde $E\left[f(s_t)\right]$ y $\mathrm{var}(f(s_t))$ son respectivamente la esperanza y la varianza de $f(s_t)$. El valor de $\rho(n)$ mide la relación entre las soluciones de la serie aleatoria y la calidad de dichas soluciones.

Es posible calcular un estimador de $\rho(n)$, $r(n)$, a partir de una serie de puntos de longitud



(a) Espacio de calidad continuo o liso.

(b) Espacio de calidad rugoso.

Figura 4.4: Representación de dos espacios de calidad uno liso o continuo y otro rugoso.

L, $(x_0, x_1, \ldots, x_L)$ como:

$$r(n) = \frac{\sum_{t=1}^{L-n} (f(s_t) - \overline{f})(f(s_{t+n}) - \overline{f})}{\sum_{t=1}^{L} (f(s_t) - \overline{f})^2} \tag{4.4}$$

donde $\overline{f} = \frac{1}{T} \sum_{t=1}^{L} f(s_t)$ y $L \gg 0$. La estimación del valor que proporciona esta medida es útil ya que una serie aleatoria de puntos o soluciones de tamaño adecuado se puede considerar representativa del espacio de calidad completo debido a que dicho espacio es estadísticamente isotrópico, es decir, en promedio, el espacio de calidad presenta las mismas características topológicas en todas las direcciones.

La *longitud de correlación* $\tau = \frac{1}{\rho(1)}$ [Schuster and Stadler, 1993,Fontana et al., 1993] mide el decrecimiento de la función de autocorrelación y la rugosidad de un espacio de calidad. A mayor longitud de correlación, más suave es la superficie sobre la que se busca.

El método de utilizar la correlación como medida de rugosidad no es nuevo y otros autores han utilizado conceptos similares con el mismo fin. Manderick [Manderick et al., 1991] y Greffenstette [Grefenstette, 1995] utilizaron la función de distribución de la calidad de un operador con el objetivo de predecir la eficacia de un algoritmo genético. La función de distribución de la calidad de un operador es una función que describe la distribución de los valores de calidad de los individuos que resultan de aplicar el operador a un individuo en función de la calidad del individuo original. En [Fogel et al., 1996], Fogel utiliza el mismo concepto anterior para determinar, durante el proceso de búsqueda, el mejor operador en términos de eficacia. En [Bornholdt, 1997], Borholdt propone un algoritmo dinámico que utiliza modelos estadísticos basados en la ampliación de los descriptores de la distribución de la calidad. Estos descriptores son, por ejemplo, la media, la varianza o los coeficientes de asimetría o curtosis. En este trabajo se realiza un estudio de distintas medidas de correlación padres-hijos que son utilizadas para determinar el modelo dinámico de búsqueda.

Los trabajos anteriores han sido desarrollados centrando sus análisis en los algoritmos genéticos. Aunque estos algoritmos comparten características de funcionamiento con otros paradigmas de algoritmos evolutivos, también tienen características diferenciadoras que modifican su comportamiento frente a las superficies de calidad. Es por este motivo que las conclusiones obtenidas en estos trabajos no pueden generalizarse a todos los tipos de AEs y sería necesario un análisis particular para cada uno de ellos.

La rugosidad de un espacio de calidad está ligada a la dificultad del mismo. Cuánto más rugoso es un espacio de calidad, el nivel de correlación entre soluciones vecinas es menor, y por lo tanto, la información que se puede extraer durante el proceso de búsqueda también es menor. Por el contrario, cuando el espacio de calidad es suave y el nivel de correlación es alto, la información que se obtiene del vecindario de las soluciones facilita la optimización de los problemas.

Los conceptos de modalidad y rugosidad están fuertemente ligados. Cuando la longitud de correlación es pequeña hay poca correlación entre soluciones vecinas y, por lo tanto, existen muchos óptimos locales. En este caso, los AEs que basan su comportamiento en vecindades son poco eficaces. En el caso contrario, cuando la longitud de correlación tiene un valor alto, indica la existencia de pocos óptimos locales. Los casos de longitud de correlación intermedia son los más frecuentes. En el trabajo desarrollado en [Stadler and Happel, 1999] se demuestra que la longitud de correlación τ, la longitud de las series aleatorias L y el número de óptimos locales están fuertemente relacionados. Si suponemos $X(x_0, \tau)$ como el conjunto de soluciones que se pueden alcanzar después de τ ejecuciones de los operadores desde x_0, la conjetura de correlación [Stadler and Schnabl, 1992] dice que existirán aproximadamente $M \approx \frac{|X|}{|X(x_0,\tau)|}$ óptimos locales. Esta conjetura se basa en la idea de que la longitud de correlación determina el diámetro de la montaña o valle más amplio de el espacio de calidad [Krakhofer and Stadler, 1996, García-Pelayo and Stadler, 1997].

Epístasis En genética, un gen se dice epistático a otro gen si la expresión fenotípica del primero depende del valor del segundo [Stickberger, 1968]. La epístasis es, por tanto, una medida de las relaciones entre los genes de un cromosoma. En el campo de la computación evolutiva, dicha medida fue introducida por Rawlins [Rawlins, 1991a]. Rawlins definió las situaciones de máxima y mínima epístasis: la situación de máxima epístasis se corresponde con situaciones en las que todos los genes son dependientes y, por el contrario, situaciones de mínima epístasis se dan cuando todos los genes son independientes. En el campo de la CE, el concepto de epístasis se relaciona con el concepto matemático de separabilidad. De aquí en adelante ambos términos se utilizarán de manera indistinta.

Matemáticamente, se define una función separable como aquella en la que no existen relaciones entre variables y cada una puede optimizarse en un proceso separado, es decir, cuando no existe epístasis. En caso contrario, la función se define como no separable y puede presentar diferentes grados de separabilidad o epístasis.

La primera definición formal de epístasis aparece en el trabajo de Davidor [Davidor, 1990a], donde se define el valor de epístasis como:

$$\epsilon(s) = f(s) - \sum_{i=0}^{l-1} \frac{1}{2^{l-1}} \sum_{\substack{t\in\Omega \\ t_i=s_i}} f(t) + \frac{l-1}{2^l} \sum_{t\in\Omega} f(t) \tag{4.5}$$

donde $\Omega = \{0,1\}^l$, esta medida solamente está descrita formalmente para espacios de codificación binaria. En el trabajo presentado en [Davidor, 1990a], se demuestra que existe una

fuerte correlación entre una epístasis alta y tiempos largos de cómputo a la hora de resolver problemas.

Relacionado con la epístasis y las dependencias entre genes de un cromosoma está el concepto de *correlación falsa o espuria* [Schaffer et al., 1990]. La correlación espuria es la falsa creencia de que dos genes están relacionados cuando en realidad no es así. Este hecho se debe, normalmente, a un tercer factor o variable denominada *variable de confusión*. La correlación espuria puede llevar al algoritmo a una pérdida de diversidad en la población y, como consecuencia, a una convergencia prematura.

Todos los trabajos citados anteriormente estudiaban la epístasis en problemas de codificación binaria. En el primer trabajo que presentó una extensión del concepto de epístasis para problemas de codificación real, los autores utilizan la distancia euclídea entre la función y su mejor aproximación polinómica para estudiar el efecto de la epístasis sobre los AGs [Rochet et al., 1996]. Posteriormente, en [Chan et al., 2003] los autores presentaron un trabajo en el cual utilizan un análisis ANOVA sobre las variables del cromosoma para estudiar la interacción entre variables y, de esta forma, estimar el nivel de epístasis. Además, utilizan la información que proporciona este análisis para modificar el operador de mutación del AG con el fin de mejorar los resultados obtenidos.

El concepto de epístasis también está relacionado con el concepto de acoplamiento (del inglés *linkage*), cuyo estudio ha cobrado gran importancia en los últimos años. Además de la publicación de libros sobre trabajos dedicados al estudio de dicho concepto en el campo de la CE [Ying-ping Chen and Meng-Hiot Lim, 2008, Ying-ping Chen, 2010], cada dos años se celebra una sesión especial dentro del congreso *CEC* dedicada exclusivamente a la presentación de trabajos sobre este tema [ssL, 2007, ssL, 2009]. La mayoría de los trabajos presentados estudian la epístasis o *linkage* de modo *on-line*, es decir, analizan las dependencias entre variables mientras resuelven el problema para adaptar los operadores de los algoritmos. Por ejemplo, en [Li et al., 2007] utilizan la información mutua entre variables para estimar las interacciones o dependencias entre variables y modificar el comportamiento de un AG dependiendo del problema que se trate de resolver. A pesar de los buenos resultados de estos trabajos, tiene mayor interés una estimación *off-line* del nivel de epístasis para utilizar esta información a la hora de seleccionar uno u otro AE.

La epístasis o separabilidad afecta a los AEs de diferente manera dependiendo del tipo de búsqueda que estos realicen. Generalmente, en las funciones con un alto grado de epístasis los algoritmos deben aprender o tratar de aprender las dependencias entre los parámetros para que la optimización se realice con éxito. Por este motivo, dichas funciones tienden a clasificarse como más complejas. Aunque, como se verá más adelante, hay algoritmos que explotan estas dependencias y, sin embargo, no son capaces de resolver funciones sin dependencias consideradas típicamente más sencillas.

Deceptividad La característica de deceptividad fue propuesta por Goldberg y se basa en la hipótesis de bloques constructivos y en el teorema de esquemas [Goldberg, 1987]. Inicialmente, este concepto estaba íntimamente relacionado con la codificación binaria, sin embargo, su idea principal puede generalizarse para cualquier codificación del espacio de calidad. Si consideramos que un esquema o hiperplano describe un área del espacio de calidad, una función se considera deceptiva cuando la calidad media del área que incluye al óptimo global es menor que la calidad media del área de algún óptimo local. En general, el concepto de deceptividad se refiere al hecho de que la función de calidad proporcione al proceso de búsqueda "pistas" erróneas, haciendo que la búsqueda se dirija hacia zonas de poca calidad

en lugar de alejarse de ellas [De Jong, 2006].

Una función es totalmente deceptiva si todos los esquemas de la función son deceptivos [Kalyanmoy, 1997]. En [Goldberg, 1989b] se clasifican las funciones en dos grupos: las funciones deceptivas de tipo I y las funciones deceptivas de tipo II. Las funciones deceptivas de tipo I son aquellas que, aunque en las primeras etapas de evolución guían al AE hacia posiciones alejadas del óptimo global, después de un número suficiente de generaciones, el AE es capaz de encontrar el verdadero óptimo de la función, mientras que esto no se cumple en el caso de las funciones deceptivas de tipo II.

En [Naudts and Verschoren, 1999], además de explicar los conceptos de epístasis y deceptividad, los autores estudiaron la relación que existe entre ellos. De dicho trabajo obtuvieron las siguientes conclusiones:

- Si la función f tiene un grado bajo de epístasis, entonces f no es deceptiva. Por lo tanto establecieron que si f es deceptiva, necesariamente tiene que tener algún grado de epístasis.
- Si f es deceptiva con un grado medio de epístasis, entonces f es deceptiva de tipo I.
- Si f es deceptiva con un grado alto de epístasis, entonces f es deceptiva de tipo II.

Por lo tanto, a la vista de estas conclusiones, los autores probaron que:

1. Si existe baja epístasis, entonces la función considerada no puede ser deceptiva.
2. Si la función es deceptiva, la epístasis permitirá diferencias entre funciones deceptivas de tipo I (epístasis media) y funciones deceptivas de tipo II (epístasis alta).

De las conclusiones obtenidas en el trabajo anterior, queda claro que la epístasis y la deceptividad de las funciones son dos características que están fuertemente ligadas.

En este apartado se han descrito las cuatro principales características que describen los espacios de calidad. Estas cuatro características son modalidad, rugosidad, epístasis y deceptividad. En la figura 4.5 se muestran las relaciones entre ellas. Tras la revisión de trabajos realizada, se ha podido concluir que los conceptos de modalidad y rugosidad están relacionados y que, de hecho, una de las principales causas de la multimodalidad de los espacios de calidad es la rugosidad. Por lo tanto, las características de modalidad y rugosidad serán analizadas de forma conjunta en el desarrollo de este trabajo. Otra de las conclusiones que se puede extraer es que la deceptividad y la epístasis está relacionadas, por lo tanto estos dos conceptos también se estudiarán de forma conjunta. Resumiendo, en una primera aproximación al problema de la caracterización topológica de los espacios de calidad, este trabajo se centrará en el análisis de dos características básicas: la modalidad y la epístasis o separabilidad. Una vez que se han establecido qué características son más relevantes, se debe tratar de describir los espacios de calidad en términos de dichas características. Previamente a la caracterización práctica, se pasa a describir las medidas de dificultad que se han tratado de establecer en el campo durante los últimos años y el porqué de su fracaso.

Medidas de caracterización topológica para espacios de calidad

Este trabajo no es el primero que trata de caracterizar el comportamiento de los AEs basándose en la topología de los espacios de calidad que trata de resolver y en la bibliografía se

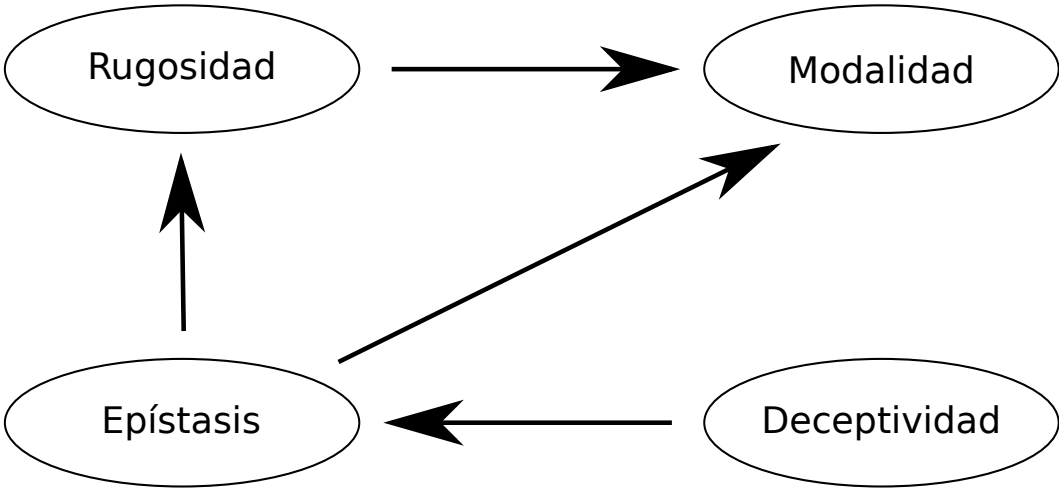


Figura 4.5: Relaciones entre las cuatro características principales de los espacios de calidad.

pueden encontrar diversos trabajos dedicados a esta tarea. Estos trabajos pretenden, también, estimar a priori cómo de adecuado es un AE para resolver un espacio de calidad con unas determinadas características. La mayor parte ellos se centran en AGs de codificación binaria. A pesar de que estos han quedado parcialmente obsoletos debido a que el interés actual se centra en problemas con codificación real, resulta interesante realizar una revisión de los mismos, sobre todo para enfatizar el problema principal de estos primeros intentos.

La Fitness Distance Correlation (FDC) es una medida de correlación entre el valor de calidad de una solución y su distancia a la solución óptima del problema. Está relacionada con los conceptos de deceptividad y rugosidad. Fue presentada en [Jones and Forrest, 1995] para estudiar el rendimiento de los AGs clásicos frente a distintas funciones de calidad. Esta medida se basa en la idea de que existe una estrecha relación entre el proceso de búsqueda de los AGs y de las heurísticas. Para realizar esta suposición, los autores se basan en que los espacios de búsqueda pueden ser vistos como grafos etiquetados y dirigidos. En el caso de los algoritmos genéticos, los nodos de los grafos estarían etiquetados según la función de calidad. En el caso de las heurísticas, las etiquetas de los nodos serían los valores de la función heurística. Un principio general de las funciones heurísticas dice que estas deberán estar bien correlacionadas con la distancia al objetivo [Doran and Michie, 1966]. Según los autores, el grado en el que la función de calidad de un AG está en concordancia con este principio da una estimación de la dificultad del espacio de búsqueda. Para el cómputo de esta medida es necesario conocer el punto óptimo de un espacio de búsqueda. Si estamos trabajando con un problema de maximización, el resultado que deseamos obtener es que el valor de calidad crezca a medida que la distancia al óptimo decrece. Por lo tanto, si tenemos una función de calidad ideal, el valor del índice de correlación será -1. Si estamos minimizando, la función de calidad ideal tendrá un índice de correlación de 1.

Al ser desarrollada para estudiar la dificultad en AGs clásicos, los ejemplos presentados en el trabajo original utilizan cromosomas de codificación binaria y distancia Hamming, ya que es la que está más relacionada con los operadores utilizados por los algoritmos genéticos. A partir de los resultados obtenidos dividieron los problemas en tres clases:

1. Problemas *misleading* o engañosos: si estamos maximizando, el índice de correlación r tiene un valor mayor o igual a 0.15. En estos problemas, la población de un AG tiene tendencia a alejarse del óptimo.

2. Problemas difíciles: cuando el valor de r está entre 0.15 y -0.15. En estos problemas hay poca correlación entre la calidad y la distancia al óptimo.

3. Problemas directos: cuando el valor de r es menor o igual que -0.15, es decir, problemas en los que la calidad tiende a crecer a medida que nos acercamos al óptimo.

En [Wang and Li, 2008], el concepto de FDC se generalizó para problemas de parámetros de codificación real. En estos casos es muy difícil o prácticamente inviable caracterizar completamente el espacio de búsqueda y obtener un valor exacto de correlación. A pesar de los buenos resultados iniciales, trabajos posteriores han demostrado que la FDC no es una medida fiable [Altenberg, 1997a, Quick et al., 1998, Kallel et al., 1999]. Concretamente, en [Altenberg, 1997a] el autor presenta un problema que un AG resuelve fácilmente pero cuyo valor de FDC indica que es un problema difícil.

Relacionadas con el concepto de epístasis han surgido numerosas medidas de dificultad. Uno de los primeros métodos presentados para analizar las relaciones entre los genes de un cromosoma son las sumas de Walsh [Heckendorn et al., 1997]. En una primera versión fueron presentadas para analizar funciones en codificación binaria y más tarde fueron extendidas para analizar funciones con otro tipo de codificación [Heckendorn and Whitley, 1999]. El método se basa en generar 2^n coeficientes, donde n es la dimensión del problema, a partir de 2^n valores de calidad. Cada coeficiente representa el valor de epístasis de cada combinación de variables. La epístasis también ha sido considerada desde el punto de vista estadístico [Reeves and Wright, 1995a,Reeves and Wright, 1995b]. En este trabajo se analizan las relaciones entre variables utilizando un análisis de varianza ANOVA. La contribución de las variables al valor de calidad se descomponen en una serie de valores denominados *efectos*. Estos pueden ser divididos en dos tipos: los efectos lineales y los efectos de interacción, los cuales representan el grado de epístasis de cada combinación de genes. Los autores de este trabajo concluyeron que no es posible determinar si una función es epistática o no basándose en una muestra del espacio de calidad aunque esta muestra se obtenga de forma controlada y no aleatoria.

La medida de varianza de epístasis presentada por Davidor [Davidor, 1990a], es la primera medida general de epístasis. Es un método flexible que permite determinar el grado de no linealidad existente en un determinado espacio de calidad. Davidor defendía que la elección de una u otra representación para los genes de los cromosomas era la causante de un mayor o menor grado de relaciones no lineales entre genes [Davidor, 1990b]. Propuso la varianza de epístasis como medida del grado de adecuación de una determinada representación a un AG para un problema determinado. Una de las desventajas de la varianza de epístasis es que ha de calcularse para el espacio de búsqueda completo, lo cual en muchas ocasiones es computacionalmente prohibitivo debido a las altas dimensiones de los problemas, sobre todo aquellos que se corresponden con aplicaciones del mundo real. Para solucionar este problema puede considerarse solamente una partición del espacio de búsqueda y realizar una estimación suponiendo que el resultado es escalable al espacio completo. Sin embargo, el propio Davidor demostró que esto no es posible [Davidor, 1990b] y que utilizar particiones del espacio de búsqueda no permite estimar la varianza de epístasis total. Al igual que ocurrió con la FDC, trabajos posteriores a la presentación de la varianza de epístasis demostraron que esta medida no es del todo fiable [Rochet et al., 1998, Reeves and Wright, 1995b]. En [Rochet et al., 1998] se presenta otra medida de epístasis, la correlación de epístasis. Esta mide de forma estadística la correlación que existe entre la función de calidad y su aproximación de primer orden sobre una muestra de puntos P del espacio de búsqueda. Si la correlación de epístasis es máxima, es decir, toma un valor de 1, entonces no existen relaciones entre los

parámetros de la función que se está analizando y, por lo tanto, dicha función no presenta ningún grado de epístasis. Tanto la varianza de epístasis como la correlación de epístasis fueron desarrolladas para detectar la presencia de epístasis en las funciones de calidad, sin embargo, ambas tienen problemas a la hora de detectar la ausencia de epístasis [Naudts, 1998]. Por ejemplo, ambas medidas proporcionan un valor cercano a 0 para funciones separables y para funciones de tipo NIaH. Sin embargo, el comportamiento de un AG en estas funciones es totalmente diferente: mientras que es capaz de resolver sin problemas las funciones separables, las funciones NIaH hacen que el rendimiento de dicho algoritmo empeore. Además de los contraejemplos presentados en el caso de la varianza de epístasis, en [Naudts, 1998] el autor demostró que las medidas de epístasis existentes no pueden estimar la dificultad de las funciones para un AG.

Otra manera de estimar la dificultad de un determinado problema es utilizando los espacios de información (*Information Landscapes*) [Borenstein and Poli, 2005]. En dicho trabajo los autores presentan una redefinición del concepto de espacio de calidad con el objetivo de utilizar la cantidad y la calidad de información disponible para predecir o estimar la dificultad de dicho espacio. Un espacio de información se define como una tripleta $(\mathcal{S}, \mathcal{V}, t)$, donde los dos primeros elementos se corresponden con los utilizados en la definición de espacio de calidad y son, respectivamente, el espacio de búsqueda y la función vecindario. El tercer elemento de la tripleta, t, se denomina función de información y para cada par de soluciones de $\mathcal{S}$, (x_i, x_j), calcula la probabilidad de que x_i sea mejor que x_j. Para medir la dificultad de un problema hay que establecer un espacio de calidad óptimo y medir la distancia entre este y el espacio de información del problema que se resuelve. Esta distancia es una estimación de la dificultad del problema. Esta medida tiene la desventaja de que es necesario conocer el óptimo de la función para realizar una estimación correcta.

La conclusión general de esta revisión de trabajos dedicados al establecimiento de medidas de dificultad de un espacio de calidad, es que estas no son aplicables a otros AEs. Es decir, que la dificultad para un AG no tiene por qué serlo para una EE. Esto quedará confirmado a lo largo del capítulo de aplicación de este trabajo, donde, exceptuando casos muy particulares, las características que para unos algoritmos resultan complejas para otros no lo son. En consecuencia, el presente procedimiento de caracterización no se basa en ninguna medida concreta de dificultad, sino que el análisis de los resultados se llevará a cabo en función de las características topológicas en sí, es decir, de la separabilidad y la modalidad del espacio.

En la siguiente sección se describirá el conjunto de funciones de prueba que será utilizado en este trabajo.

4.1.2 Conjunto de funciones de prueba

El componente principal de un espacio de calidad es la función de aptitud o calidad que lo define. Para realizar la caracterización de un algoritmo, en el campo de la optimización, se utilizan conjuntos de funciones de prueba. Estos conjuntos incluyen funciones que presentan diferentes características intentando cubrir un amplio rango de las mismas para probar las capacidades de los algoritmos a la hora de resolverlas y, de esta forma, caracterizar su comportamiento.

En este trabajo se utiliza un conjunto de funciones formado por aquellas que han sido más utilizadas en trabajos de optimización con AEs [Yao et al., 1999, Dixon and Szegö, 1978, Ali et al., 2005] en la última década. Su expresión matemática y sus características principales,

como límites de los parámetros, posición y valor del óptimo, se especifican en el apéndice B. Este conjunto incluye funciones escalabes y no escalables en dimensión. En cuanto a las características topológicas, se han incluido funciones que representan espacios de calidad suaves y otros con distintos niveles de rugosidad. También se han incluido espacios de calidad con amplias superficies planas sin información de gradiente que pueda guiar a los algoritmos hacia la solución.

Las características básicas de estas funciones se muestran en la tabla 4.1. Estas son las propiedades que generalmente se encuentran en la bibliografía sobre estas funciones. Sin embargo, como veremos en secciones posteriores, no son suficientes para estimar la topología del espacio de calidad. Ya que, por ejemplo, dentro de las funciones multimodales existen diferentes tipos de modalidad y lo mismo ocurre con las funciones unimodales y con la característica de separabilidad.

Dimensionalidad

En la tabla 4.1, además de las características de modalidad y separabilidad, aparece una tercera columna que indica la dimensión del problema, en otras palabras, el número de parámetros de cada función. Como se dijo al comienzo de este apartado, algunas de las funciones utilizadas tienen dimensión fija. Estas funciones se corresponden con el llamado conjunto de prueba de Dixon-Szegö [Dixon and Szegö, 1978]. Son interesantes desde el punto de vista de los AEs debido a sus características topológicas y a que, en la mayor parte de los casos, no son simétricas. Sin embargo, desde el punto de vista de la dimensionalidad no son especialmente complejas ya que el número de parámetros de estas funciones es bajo, entre 2 y 6 parámetros.

El resto de funciones son escalables en cuánto a dimensionalidad y serán utilizadas para comparar el rendimiento de los AEs al variar el número de parámetros de las funciones. La dimensionalidad de los problemas afecta a los algoritmos en distinta medida empeorando el rendimiento de los mismos, en algunos casos de manera exponencial. Este problema se conoce en la bibliografía como la maldición de la dimensionalidad [Tang et al., 2007] (del inglés, *curse of dimensionality*).

Al aumentar la dimensión del problema, la complejidad del problema también crece, debido a que existen más variables a optimizar. El aumento de la dimensionalidad afecta tanto a la modalidad como a la separabilidad. En las funciones multimodales el número de óptimos locales generalmente crece de forma exponencial con la dimensión como se indica en [Yao et al., 1999]. Además, el espacio de soluciones de un problema crece a medida que aumentan las dimensiones y es necesaria una estrategia de búsqueda eficiente para explorar el espacio de calidad que define dicha función. En cuanto a la separabilidad, en el caso de las funciones no separables, a mayor número de parámetros más dependencias entre los mismos y, por lo tanto, los algoritmos deberán aprender más relaciones con las complicaciones que ello conlleva.

En este trabajo se utilizan tres dimensiones diferentes para las funciones escalables: baja, media y alta, que se corresponden con 10, 30 y 50 respectivamente. Estos valores son los que se utilizan normalmente en las competiciones de AEs (véase [Suganthan et al., 2005, Liang et al., 2006, Huang et al., 2007, Li et al., 2008, Zhang et al., 2008, Mallipeddi and Suganthan, 2010]).

Una vez presentado el conjunto de funciones prueba que se utilizará en este trabajo y sus características básicas, a continuación se explicará como han sido desarrollados los al-

Función	Dimensión	Separabilidad	Modalidad
Axis Parallel Hyperellipsoid	10, 30, 50	L-Separable	Unimodal
Schwefel 2.22	10, 30, 50	L-Separable	Unimodal
Sphere Model	10, 30, 50	L-Separable	Unimodal
Step	10, 30, 50	L-Separable	Unimodal
Sum of Different Power	10, 30, 50	L-Separable	Unimodal
Colville	4	No-Separable	Unimodal
Easom	2	No-Separable	Unimodal
Kowalik's	4	No-Separable	Unimodal
Matyas	2	No-Separable	Unimodal
Perm	10, 30, 50	No-Separable	Unimodal
Schwefel 1.2	10, 30, 50	No-Separable	Unimodal
Schwefel 2.21	10, 30, 50	No-Separable	Unimodal
Zakharov	10, 30, 50	No-Separable	Unimodal
Ackley	10, 30, 50	L-Separable	Multimodal
Aluffi-Pentini's	2	L-Separable	Multimodal
Becker and Lago	2	L-Separable	Multimodal
Bohachevsky 1	2	L-Separable	Multimodal
Cosine Mixture	10, 30, 50	L-Separable	Multimodal
Rastrigin	10, 30, 50	L-Separable	Multimodal
Schwefel	10, 30, 50	L-Separable	Multimodal
Beale	2	No-Separable	Multimodal
Bohachevsky 2	2	No-Separable	Multimodal
Dekkers and Aarts	2	No-Separable	Multimodal
Goldstein Price	2	No-Separable	Multimodal
Griewank	10, 30, 50	No-Separable	Multimodal
Hartman 3	3	No-Separable	Multimodal
Hartman 6	6	No-Separable	Multimodal
Levy	10, 30, 50	No-Separable	Multimodal
Penalized 1	10, 30, 50	No-Separable	Multimodal
Penalized 2	10, 30, 50	No-Separable	Multimodal
Rosenbrock	10, 30, 50	No-Separable	Multimodal
Shekel Family 5	4	No-Separable	Multimodal
Shekel Family 7	4	No-Separable	Multimodal
Shekel Family 10	4	No-Separable	Multimodal
Shekel's Foxholes	2	No-Separable	Multimodal
SixHump Camel Back	2	No-Separable	Multimodal

Tabla 4.1: Listado de funciones utilizadas en esta tesis y sus características básicas.

goritmos de caracterización que serán utilizados para extraer más información en cuanto a la topología de estas funciones.

4.1.3 Algoritmos de caracterización

En la sección 4.1.1 han sido definidas las características de los espacios de calidad más estudiadas en el campo de la CE para llegar a la conclusión de que, en una primera aproximación, existen dos de especial relevancia a la hora de determinar la topología de un espacio de calidad: la separabilidad y la modalidad. Los conceptos de separabilidad y modalidad no son nuevos y, de hecho, en las competiciones sobre AEs las funciones de los conjuntos de prueba se clasifican según estas características. Sin embargo, dicha clasificación se realiza *grosso modo* y de forma binaria, es decir, se distingue entre funciones separables o no separables y funciones unimodales o multimodales, sin tener en cuenta que pueden existir diferentes tipos y grados de separabilidad y modalidad. En el apartado anterior también se ha realizado una revisión de trabajos en los cuales se proponen medidas de estimación de la topología de los espacios de calidad en términos de estas dos características. Entre las desventajas de las medidas utilizadas destaca el hecho de que, en la mayor parte de ellas, es necesario conocer la expresión analítica de la función de calidad para realizar el análisis. Además, estas medidas han sido desarrolladas para analizar el comportamiento de los AGs sin tener en cuenta otros paradigmas o modelos con procesos de búsqueda muy diferentes a los de un AG. Por último, en la mayoría de los casos se centran en estudiar espacios de calidad binarios, lo cual no permite una generalización de estas medidas al análisis de espacios de calidad de problemas reales.

Por lo tanto, surge la necesidad de desarrollar nuevos algoritmos que permitan analizar en mayor profundidad la topología de los espacios de calidad en términos de separabilidad y modalidad. En esta sección se presentan tres algoritmos de análisis: uno de ellos para el análisis de separabilidad y dos para analizar la modalidad de los espacios de calidad.

Separabilidad

Notación que será utilizada en esta sección:

- $f : \mathbb{R}^n \longrightarrow \mathbb{R}$, es una función de codificación real.
- $x \in \mathbb{R}^n$, es un vector de parámetros reales.
- x_i, es el elemento de índice i del vector x, donde $0 \leq i < n$
- $f(x : i : r)$, es el resultado de aplicar la función f sobre el vector x que en la posición i toma el valor de $r \in \mathbb{R}$.
- $\delta(f, x_1, x_2, r, i)$, es el aporte a la calidad de la variable con índice i en la función f, dados dos puntos del espacio de búsqueda de f, x_1 y $x_2 \in \mathbb{R}^n$ y un valor $r \in \mathbb{R}$.

Definición matemática de separabilidad El concepto de separabilidad está relacionado con los conceptos de deceptividad y epístasis y hace referencia a las dependencias que presentan los parámetros de las funciones y cómo afectan estas dependencias a la calidad de los individuos. Las funciones separables son aquellas en las que no existen relaciones o interdependencias entre las variables. Por lo tanto, una función separable $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ puede ser

optimizada mediante n procesos independientes, sobre cada una de las variables x_i ignorando los valores del resto de variables. Suponiendo que el proceso de optimización consista en la minimización de una función, el concepto de separabilidad puede expresarse de la siguiente forma [Hansen, 2007]:

$$argmin_{x_1} f(x_1, \dots), \dots, argmin_{x_n} f(\dots, x_n) = argmin_{x_1,\dots,x_n} f(x_1, \dots, x_n) \tag{4.6}$$

es decir, el resultado de minimizar cada variable por separado ($argmin_{x_i} f(x_i, \dots)$ es el resultado de minimizar la variable x_i) es igual al resultado de minimizar todas las variables de forma conjunta ($argmin_{x_1,\dots,x_n} f(x_1, \dots, x_n)$).

Típicamente, las funciones separables son funciones lineales, sin embargo, existen funciones separables que pueden presentar no linealidades a la hora de evaluar la contribución de cada variable a la calidad total [Whitley et al., 1995]. A pesar de estas no linealidades, en este tipo de funciones, el valor óptimo de cada parámetro también se puede determinar de forma independiente al resto de variables. En este trabajo se distinguirán ambos tipos de funciones. Las primeras, aquellas que no presentan ningún tipo de no linealidad, se denominarán funciones linealmente separables o *L-separables* [Mosk-Aoyama and Shah, 2006]. Aquellas que sí presentan no linealidades aunque pueden optimizarse en procesos separados recibirán el nombre de funciones no linealmente separables o *NL-separables* [Bäck et al., 1997].

Generalmente, las funciones de optimización que representan problemas de la vida real son no separables, es decir, los parámetros están relacionados y la bondad del valor de un parámetro depende de los valores de otros parámetros. En las últimas décadas, muchos autores han hecho esfuerzos para desarrollar AEs capaces de lidiar con funciones no separables [Hansen and Ostermeier, 1996, Auger and Hansen, 2005], básicamente tratando de aprender las dependencias entre parámetros durante el proceso de optimización. Sin embargo, aunque este procedimiento ha proporcionado resultados satisfactorios, los AEs desarrollados tienen tendencia a presentar resultados comparativamente peores en funciones separables. Este problema hace que se tienda a clasificar las funciones utilizadas en los conjuntos de prueba en términos de separabilidad.

Resumiendo, en el marco de este trabajo se consideran tres tipos de separabilidad:

- **Funciones linealmente separables (L-Separables):** son funciones entre cuyas variables no existe ningún tipo de dependencia. Debido a esto, dichas funciones pueden separarse en n funciones de 1 variable independientes y optimizar cada una de estas funciones por separado.

- **Funciones no linealmente separables (NL-Separables):** son funciones entre cuyas variables existen dependencias de tipo no lineal. A pesar de estas dependencias, cada variable puede optimizarse en un proceso independiente.

- **Funciones no separables (No-Separables):** en este tipo de funciones existen dependencias de diferentes grados entre variables.

Algoritmo de caracterización de espacios de calidad en términos de separabilidad Como se explicó en la sección anterior, en una función separable no existen dependencias entre variables mientras que en una función no separable o epistática estas dependencias existen en diferentes grados de magnitud. A primera vista, parece obvio que una función separable será más sencilla de resolver que una función no separable ya que las funciones separables pueden dividirse en n funciones de una variable que pueden ser optimizadas en n procesos de

optimización diferentes de una única variable. En el caso de las funciones no separables, al menos dos parámetros de la función interaccionan entre sí, por lo tanto, durante el proceso de optimización esos dos parámetros deben ser modificados de forma simultánea [Salomon, 1997].

En una primera aproximación, el análisis de la separabilidad de las funciones de calidad se realiza derivando las funciones con respecto a cada una de las variables del problema y comprobar si las derivadas dependen de otras variables que no sea la variable de derivación. Formalmente, una función será linealmente separable si cumple:

$$\forall i : 0 \leqslant i < n, \frac{\partial f}{\partial x_i} = cte. \tag{4.7}$$

donde n es la dimensión del problema.

A pesar de la sencillez de este método, no aporta más información que la división binaria entre funciones linealmente separables y funciones que no lo son. Aunque esta información inicialmente es útil, no proporciona datos suficientes para determinar el grado de separabilidad de una función. Además para determinar la separabilidad de la función se utiliza la expresión analítica de la misma, que, como ya se ha mencionado anteriormente, en problemas de aplicaciones reales muchas veces no es posible conocer.

Con el objetivo de clasificar claramente las funciones que se desean resolver, en este trabajo se presenta un algoritmo de caracterización de funciones en términos de separabilidad. Este algoritmo de caracterización se basa en la idea de dependencia e independencia entre variables y cómo afectan las variables del problema al aporte a la calidad de otra de las variables sin necesidad de conocer en ningún momento la expresión analítica de la función objetivo. El grado separabilidad de una función puede determinarse si conocemos en qué medida el aporte de calidad de una variable se ve afectado por el resto de las variables. Por lo tanto, se debe analizar el aporte a la calidad de una variable en función de los valores que tomen el resto de las variables. Si este aporte es el mismo cualesquiera que sean los valores del resto de las variables, estas son independientes y por lo tanto la función será separable, en caso contrario la función será no separable y deberemos determinar el grado de separabilidad como una medida de cómo afectan el resto de las variables a la variable analizada.

Análisis preliminar: Aporte a la calidad. A continuación se explicará el concepto de aporte de calidad de una variable y como este influye o no en el resto de variables, este concepto es necesario para analizar problemas en términos de separabilidad.

Dada una función $f : \mathbb{R}^n \to \mathbb{R}$, dos puntos $x_1, x_2 \in \mathbb{R}^n$ del espacio de búsqueda de la función f, un valor aleatorio r y un índice de variable $i, 0 \leq i < n$; en este trabajo se define el aporte a la calidad δ como el valor con el cual contribuye la variable de índice i al valor de calidad $f(x)$, calculado según:

$$\delta(f, x_1, x_2, r, i) = f(x_1 : i : r) - f(x_2 : i : r) \tag{4.8}$$

donde $f(x : i : r)$ es el valor de la función f cuando se aplica al punto x cuando la variable de índice i toma el valor de r.

Con el objetivo de analizar si el aporte a la calidad de la variable i es dependiente del resto de las variables de la función, un usuario deberá seguir los siguientes pasos:

- **Entrada:**

- Dos puntos del espacio de búsqueda generados de forma aleatoria, $x_1, x_2 \in \mathbb{R}^n$.
- El índice i de la variable a analizar.
- Dos valores aleatorios r_1 y $r_2 \in \mathbb{R}$.

1. En los puntos $x_1, x_2 \in \mathbb{R}^n$ se fijan los valores de las $n-1$ variables cuyo índice no sea i.
2. La variable de índice i toma los valores r_1 y r_2.
3. Se calculan los valores $\delta_1(f, x_1, x_2, r_1, i)$ y $\delta_2(f, x_1, x_2, r_2, i)$.
4. Se comparan dichos valores, si son iguales la variable con índice i es independiente del resto de variables, si los valores son diferentes, entonces la variable es dependiente.

Para analizar el grado de separabilidad a partir de los valores calculados según la definición del concepto de aporte a la calidad se parte de las siguientes premisas:

- Si no existen dependencias entre las variables de un problema, el aporte a la calidad de la variable i es independiente del valor que tomen el resto de las variables del problema. Por lo tanto, variando el valor de la variable i y fijando el valor de las *n-1* variables restantes, el aporte a la calidad de i será el mismo.
- Si existen dependencias entre las variables de un problema, el aporte a la calidad de la variable i depende del valor que tomen el resto de las variables del problema. Entonces, si se modifica el valor de la variable i y no se cambia el valor de las *n-1* variables restantes, el aporte a la calidad de i será diferente.

Con el objetivo de explicar de manera práctica el concepto de aporte a la calidad y su posterior aplicación en el análisis de separabilidad de las funciones, el algoritmo anterior ha sido aplicado a tres funciones conocidas en el campo de la optimización. Estas tres funciones son la función *Rastrigin* (ecuación 4.9), la función *Ackley's* (ecuación 4.10) y la función *Hartman6* (ecuación 4.11). Su definición matemática es la siguiente:

$$f_{Rast}(x) = \sum_{i=1}^{n} (x_i^2 - 10.0cos(2.0\pi x_i + 10.0)) \tag{4.9}$$

$$\begin{aligned} f_{Ack}(x) = -20.0\exp\left(-0.2 * \sqrt{\frac{1.0}{n}\sum_{i=1}^{n} x_i^2}\right) \\ -\exp\left(\frac{1.0}{n}\sum_{i=1}^{n} cos(2.0\pi x_i)\right) + 20.0 - e \end{aligned} \tag{4.10}$$

$$f(x)_{Hart} = -\sum_{i=1}^{6} c_i \exp\left[\sum_{i=1}^{6} a_{ij}(x_i - p_{ij})^2\right] \tag{4.11}$$

Supongamos que trabajamos con las tres funciones anteriores en problemas de dimensión 6, es decir, 6 variables por cromosoma (este valor ha sido escogido debido a que la función

Hartman6 no es escalable y está definida para dimensión 6. De esta forma utilizamos los mismos puntos para todas las funciones) y se quiere analizar si el aporte a la calidad de la variable con índice *0* es dependiente del resto de las variables.

Para ello se siguen los siguientes pasos:

1. Se generan dos puntos fijos del espacio de búsqueda (x_1 y x_2), en los cuales se fijan los valores de las *5* variables, en este caso dichos puntos son:

$$x_1 = \{-0.92, -0.17, 0.48, r, 0.15, 0.43\}$$
$$x_2 = \{-0.23, 0.75, 0.44, r, -0.44, 0.28\}$$

2. Calculamos el aporte a la calidad de la variable *0* utilizando los puntos x_1 y x_2 y como valores de la variable de índice *3*: $r_1 = 0.40$ y $r_2 = 0.64$.

Los resultados de estos cálculos se presentan en la tabla 4.2. A la vista de los resultados obtenidos en este análisis simple, podemos concluir que el aporte a la calidad de la variable *3* en la función *Rastrigin* es independiente del valor que tomen el resto de las variables del problema, por lo tanto es una función separable. Sin embargo, en el caso de las función *Ackley's* y *Hartman6* el aporte a la calidad de la variable *3* es diferente según tomen valores el resto de las variables del problema, por lo tanto existen algún tipo de no separabilidad o dependencia entre las variables del problema.

Función	δ_1	δ_2
Rastrigin	1.132	1.132
Ackley's	0.828	0.714
Hartman6	-0.087	-0.051

Tabla 4.2: Resultados del análisis de aporte a la calidad

Aunque este análisis es sencillo, no es posible determinar el grado de separabilidad de una función analizando únicamente dos puntos y una variable, sino que es necesario realizar un análisis más exhaustivo considerando más puntos fijos y todas las variables del problema. Dependiendo de la precisión de dicha estimación que se desee obtener en la clasificación será necesario analizar más o menos variables y un mayor o menor número de puntos. Debido a esta complejidad computacional, el algoritmo que se presenta en este trabajo pretende realizar una estimación del grado de separabilidad y no un análisis exhaustivo. La precisión de la estimación dependerá de los recursos disponibles.

En una primera aproximación, se ha desarrollado un método gráfico para analizar la separabilidad de la funciones. Dicho método se explica a continuación.

Método gráfico Para generar las gráficas bi-dimensionales que permiten analizar el tipo y grado de separabilidad de una función se deben seguir los siguientes pasos:

- **Entrada:**
  - Número de muestras por variable N. Se generan de manera aleatoria en el rango $[x_{low}, x_{up}]$, donde x_{low} y x_{up} representan, respectivamente, los límites inferior y superior de cada variable del problema.

- Número de puntos fijos P para los cuales se generaran las gráficas. Cada punto $p \in P$ es de n dimensiones, donde n es la dimensión del problema, y se genera aleatoriamente.

1. Para cada dimensión i del problema se generan P gráficas bi-dimensionales de la siguiente forma:
   1.1. Para cada punto p del conjunto de puntos aleatorios P se fijan las $n-1$ variables (todas excepto i).
   1.2. La variable i toma los valores generados en conjunto N y se calcula el valor de calidad para cada punto y el valor de calidad del punto fijo.
   1.3. Se calcula el aporte a la calidad de cada punto del conjunto.
   1.4. Con los N puntos se "pinta" una gráfica de aporte a la calidad.
2. Se comparan las P gráficas por punto fijo y variable para determinar su grado de separabilidad.

Ejemplos de aplicación del método gráfico. A continuación se ilustra el funcionamiento de este método sobre las tres funciones que fueron utilizadas en el cálculo del aporte a la calidad, es decir, la función *Rastrigin*, la función *Ackley's* y la función *Hartman 6*.

Para cada una de las funciones se ha aplicado el algoritmo anterior muestreando cada variable con 1000 puntos aleatorios del espacio de búsqueda. El número de puntos fijos utilizados es 10 y, al igual que en el análisis previo de aporte a la calidad, el número de variables utilizada en los problemas es 6.

En las figuras 4.6, 4.7, 4.8, 4.9, 4.10 y 4.11 se muestran las gráficas obtenidas tras la aplicación de este algoritmo sobre cada una de las funciones. En dichas gráficas, el eje de abscisas representa el valor que toma la variable analizada normalizado en el rango [-1:1]. En las gráficas de las figuras 4.6, 4.8 y 4.10 el eje de ordenadas se corresponde con el valor de calidad de la función y se representan las 10 gráficas que se obtienen para cada punto fijo. En las gráficas de las figuras 4.7, 4.9 y 4.11 se representa el aporte a la calidad de cada uno de los 1000 puntos r estudiados (eje de abscisas) tomando como referencia uno de los puntos fijos, siempre el mismo. Es decir, representa los valores $\delta(f, x_0, x_i, r)$ donde x_i es cada uno de los puntos fijos utilizados para el análisis y r son 1000 puntos aleatorios en el rango $[-1.0, 1.0]$. Para el análisis de las gráficas nos basamos en el concepto de aporte de calidad explicado en esta misma sección.

En el caso de la función *Rastrigin* (ver figuras 4.6 y 4.7), las gráficas para cada punto fijo son paralelas, esto se debe a que el aporte a la calidad de cada punto r analizado (eje de abscisas) es el mismo para cualquiera de ellos. Esto se demuestra en las gráfica de la figura 4.7, donde puede verse que el aporte a la calidad de cada punto es constante en todos ellos. Es decir, este aporte es independiente del valor que tomen el resto de las variables y, debido a esta independencia, puede afirmarse que esta función es separable.

En el caso de las funciones *Ackley's* y *Hartman6* el resultado es diferente, como puede observarse en sus gráficas correspondientes (figuras 4.8, 4.9, 4.10 y 4.11). El aporte a la calidad de las variables analizadas varía dependiendo del valor que tomen el resto de las variables y, además, la influencia es distinta dependiendo de cada variable, lo cual se deduce de las diferentes "formas" de cada gráfica en cada una de las variables. Por lo tanto, se puede afirmar que estas funciones tienen algún tipo de no separabilidad.

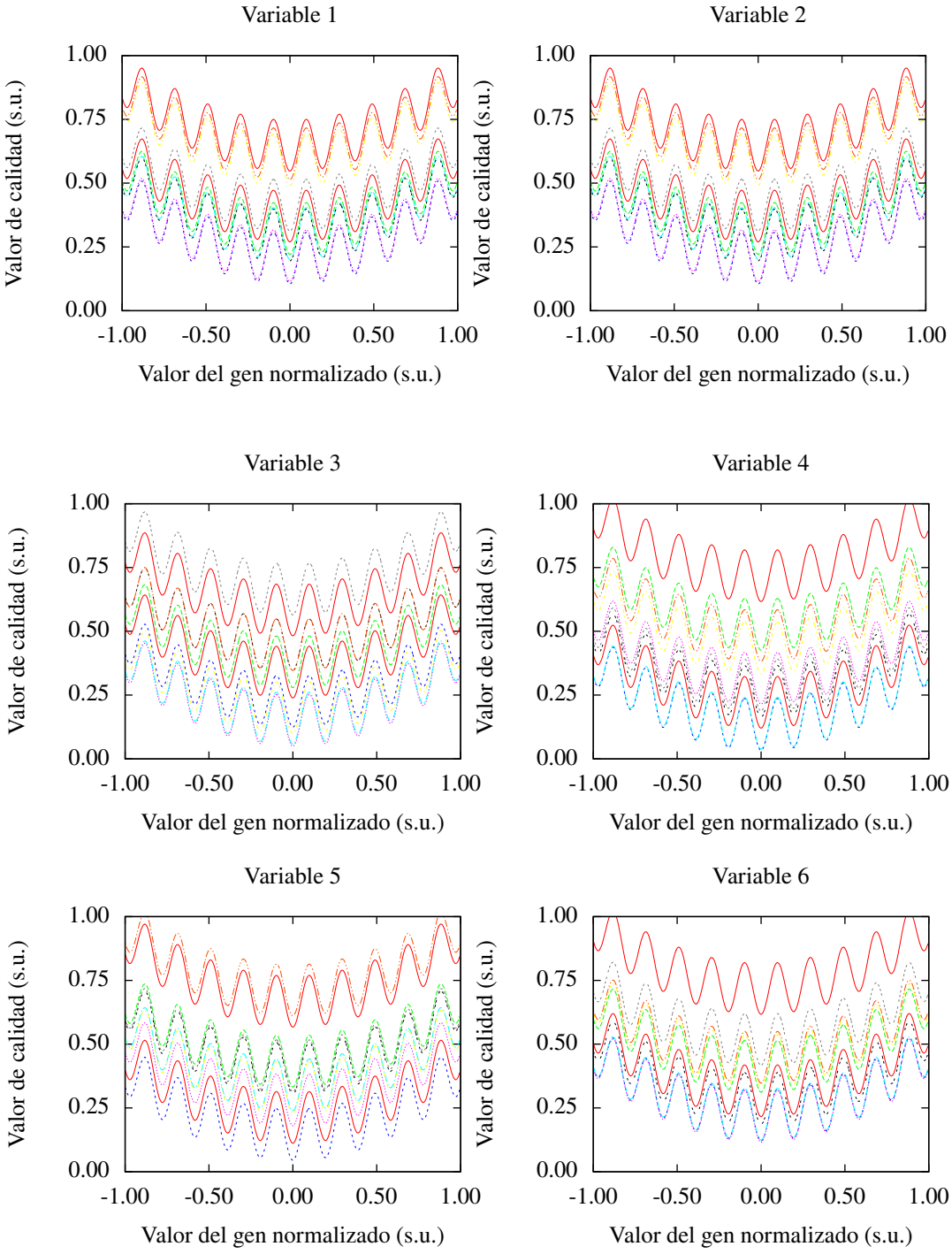


Figura 4.6: Análisis gráfico de separabilidad para la función *Rastrigin*. De izquierda a derecha y de arriba a abajo se representan las 6 variables analizadas. Cada línea de color se corresponde con el análisis de un punto fijo y un barrido aleatorio en la variable indicada en el título. El eje de abscisas representa el valor de las variables normalizado al rango [-1:1] y el eje de ordenadas representa la calidad de cada variable según el punto fijo analizado.

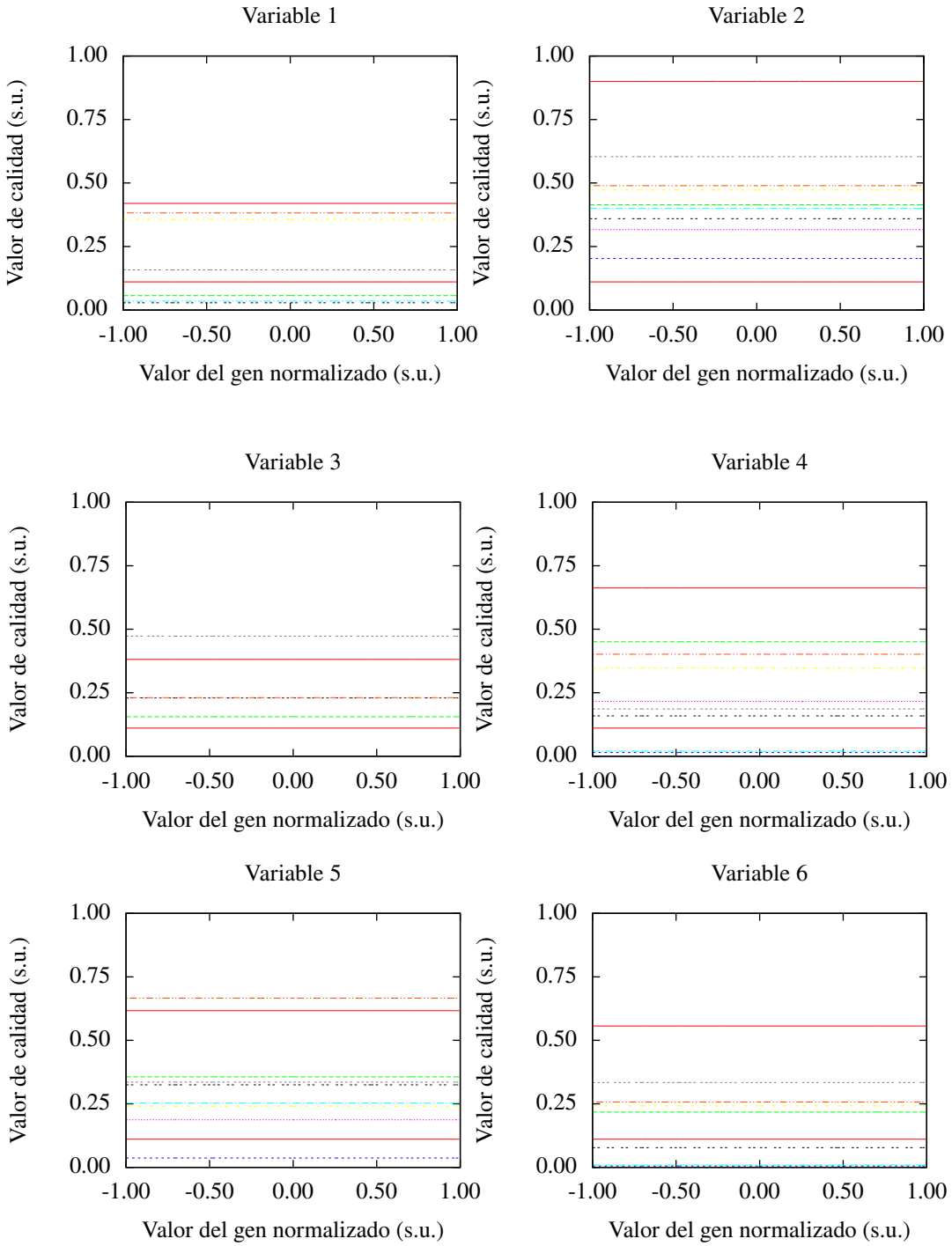


Figura 4.7: Análisis gráfico de separabilidad para la función *Rastrigin*. De izquierda a derecha y de arriba a abajo se representan las 6 variables analizadas. Cada línea de color se corresponde con el análisis de un punto fijo y un barrido aleatorio en la variable indicada en el título. El eje de abscisas representa el valor de las variables normalizado al rango [-1:1] y el eje de ordenadas representa el aporte a la calidad de cada variable según el punto fijo analizado.

Si nos fijamos con más detalle en las gráficas de ambas funciones, podemos observar otra característica que también debe ser analizada. Esta característica es la posición (coordenada *x*) del punto óptimo, es decir, el punto con menor valor de calidad, de cada variable estudiada. Esta característica debe analizarse variable a variable. En el caso de la función *Ackley's* (ver las figuras 4.8 y 4.9) en cada una de las gráficas representadas en la figura podemos ver que el punto óptimo de cada línea (punto fijo) está en la misma posición *x*. De esta característica podemos deducir que optimizando cada variable en un proceso independiente llegaríamos a la misma solución óptima. Por lo tanto, esta función tiene cierto grado de separabilidad. Sin embargo, cuando el proceso de optimización se hace a la vez con todas las variables, la ``forma" o topología de la superficie de calidad es diferente dependiendo de los valores que tome cada variable y, por lo tanto, existe también cierto grado de dependencia o no separabilidad entre las variables.

Este grado de dependencia modifica la topología de la superficie de calidad pero no la posición del óptimo. Esta característica permitiría optimizar cada variable mediante procesos independientes. Sin embargo, si utilizamos AEs, los cuales no permiten separar los procesos de optimización, cambios en una variable provocan cambios en la superficie de búsqueda que, como consecuencia, afectarían al proceso de búsqueda de los algoritmos, por lo tanto la función *Ackley's* es de tipo *NL-Separable* [Whitley et al., 1995].

En el caso de la función *Hartman 6* (ver figura 4.10 y 4.11), el punto óptimo de cada variable por cada punto fijo está situado en una coordenada *x* diferente. Debido a esto no se puede realizar un proceso de optimización separando variables, ya que dependiendo del valor que tomen algunas de ellas la posición del óptimo para cada una de las estudiadas será diferente. Por lo tanto, la función *Hartman 6* es *No-Separable* ya que no presenta ningún tipo de separabilidad y no es posible realizar un proceso de optimización para cada variable de forma independiente.

Por lo tanto, el algoritmo propuesto es una aproximación que permite obtener más información acerca de las funciones de calidad aparte de la clasificación típica entre funciones separables y funciones no separables. En primer lugar, permite clasificar las funciones en tres tipos dependiendo de su grado de separabilidad: L-Separables, NL-Separables y No-Separables. Además, al analizar las funciones variable a variable permite estimar qué variables son dependientes y cuáles no y el grado de dependencia entre ellas. Como se verá en la sección de aplicación, la información adicional que proporciona este algoritmo no solo es útil para clasificar las funciones, sino que es de gran relevancia a la hora de caracterizar los AEs.

Resultados del análisis de separabilidad Este método ha sido aplicado a las funciones de prueba seleccionadas en este trabajo (ver apéndice B) con el objetivo de clasificarlas en cuanto a términos de separabilidad. Para realizar la estimación del grado de separabilidad se utilizó el algoritmo explicado en la sección 4.1.3. Los parámetros utilizados fueron los siguientes:

- N = 1000, es decir, se analizan 1000 muestras por cada par variable-punto fijo.
- P = 10, se utilizaron 10 puntos fijos del espacio de búsqueda.

Con este nivel de precisión se obtuvieron los resultados que se presentan en la tabla 4.3 donde se clasifican las funciones en *Separables*, *NL-Separables* y *No-Separables*. Como se puede observar en esta tabla, el número de funciones *No-Separables* es mayor que el número de funciones *Separables* siguiendo con las indicaciones dadas en [Whitley et al., 1996, Whitley

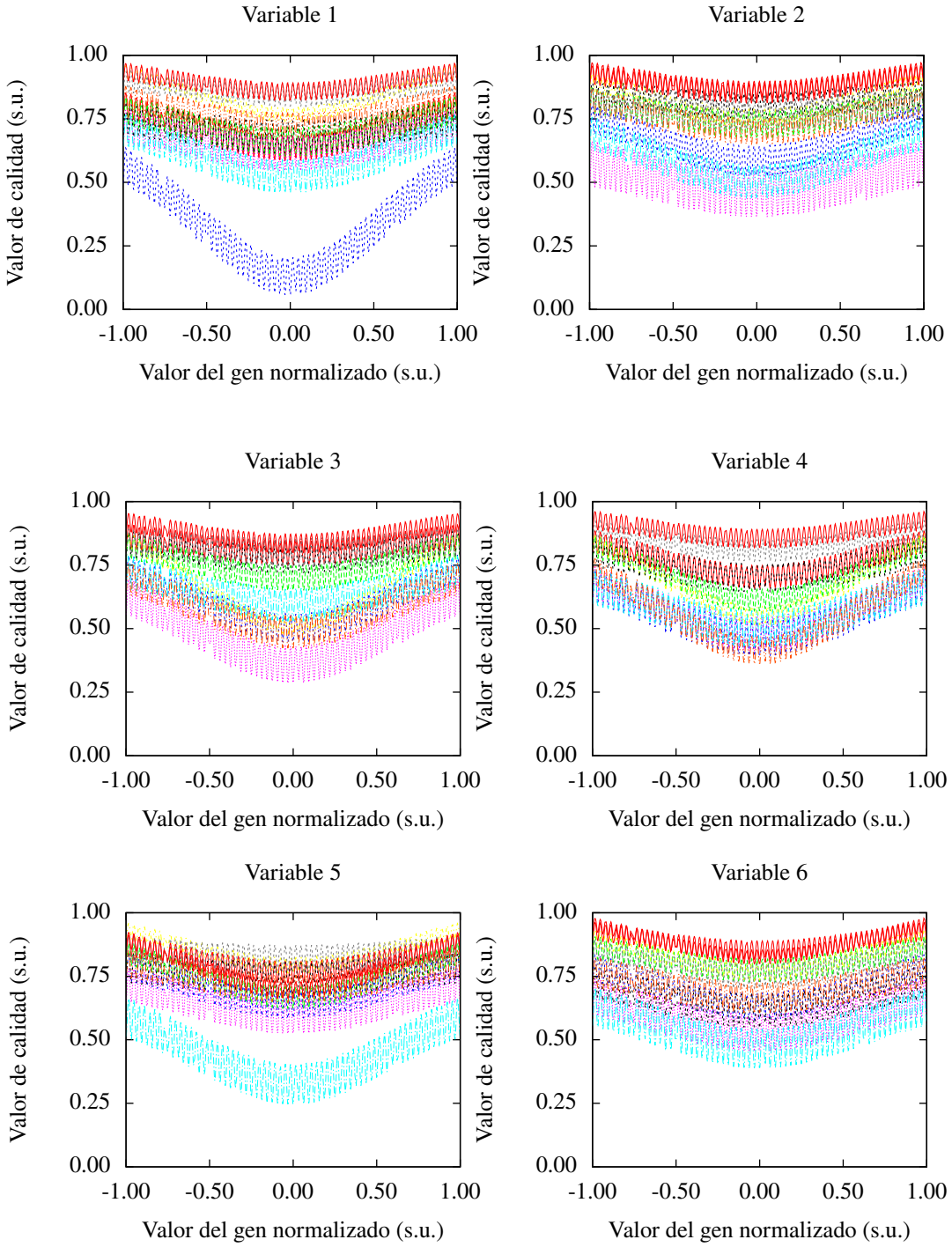


Figura 4.8: Análisis gráfico de separabilidad para la función *Ackley's*. De izquierda a derecha y de arriba a abajo se representan las 6 variables analizadas. Cada línea de color se corresponde con el análisis de un punto fijo y un barrido aleatorio en la variable indicada en el título. El eje de abscisas representa el valor de las variables normalizado al rango [-1:1] y el eje de ordenadas representa la calidad de cada variable según el punto fijo analizado.

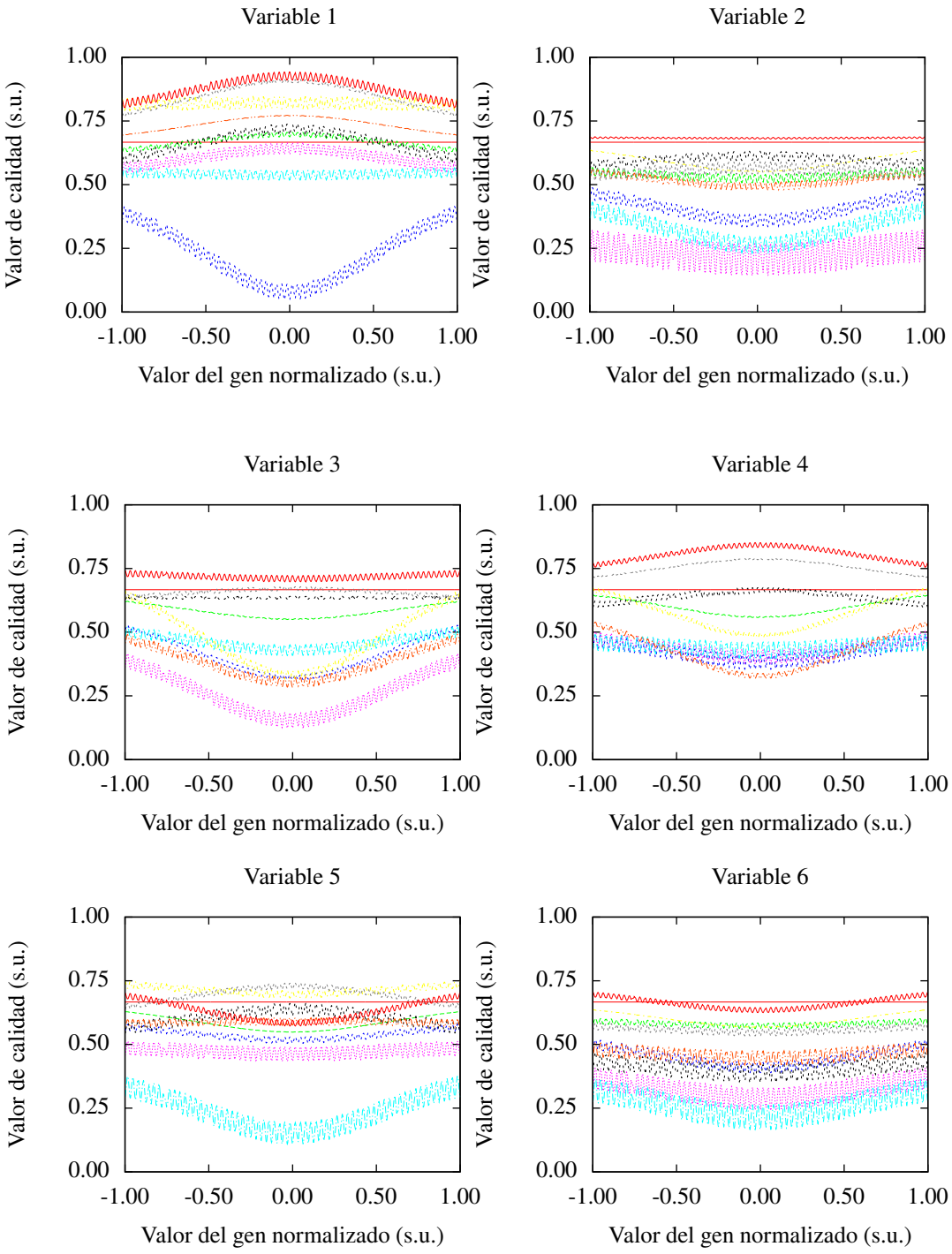


Figura 4.9: Análisis gráfico de separabilidad para la función *Ackley's*. De izquierda a derecha y de arriba a abajo se representan las 6 variables analizadas. Cada línea de color se corresponde con el análisis de un punto fijo y un barrido aleatorio en la variable indicada en el título. El eje de abscisas representa el valor de las variables normalizado al rango [-1:1] y el eje de ordenadas representa el aporte a la calidad de cada variable según el punto fijo analizado.

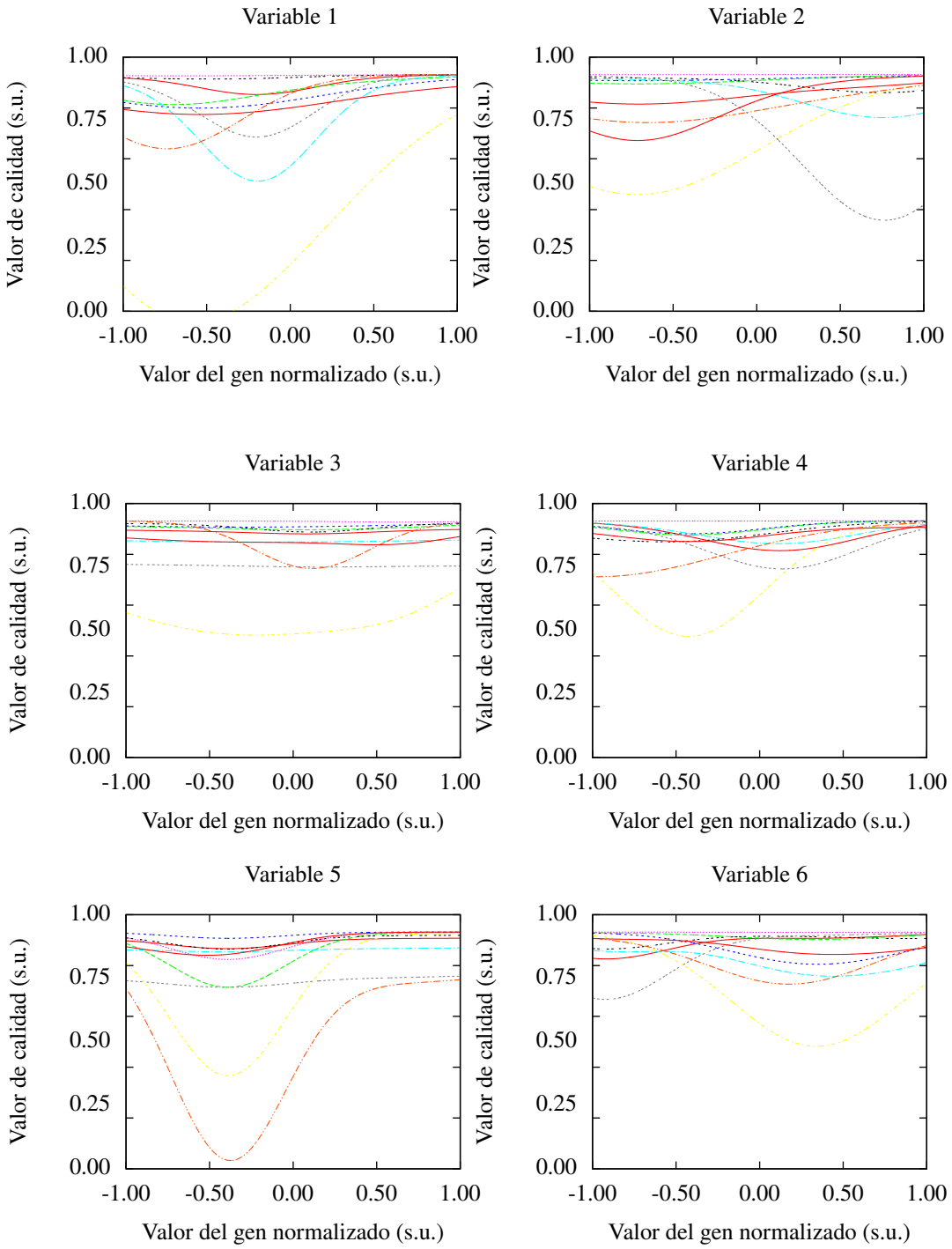


Figura 4.10: Análisis gráfico de separabilidad para la función *Hartman 6*. De izquierda a derecha y de arriba a abajo se representan las 6 variables analizadas. Cada línea de color se corresponde con el análisis de un punto fijo y un barrido aleatorio en la variable indicada en el título. El eje de abscisas representa el valor de las variables normalizado al rango [-1:1] y el eje de ordenadas representa la calidad de cada variable según el punto fijo analizado.

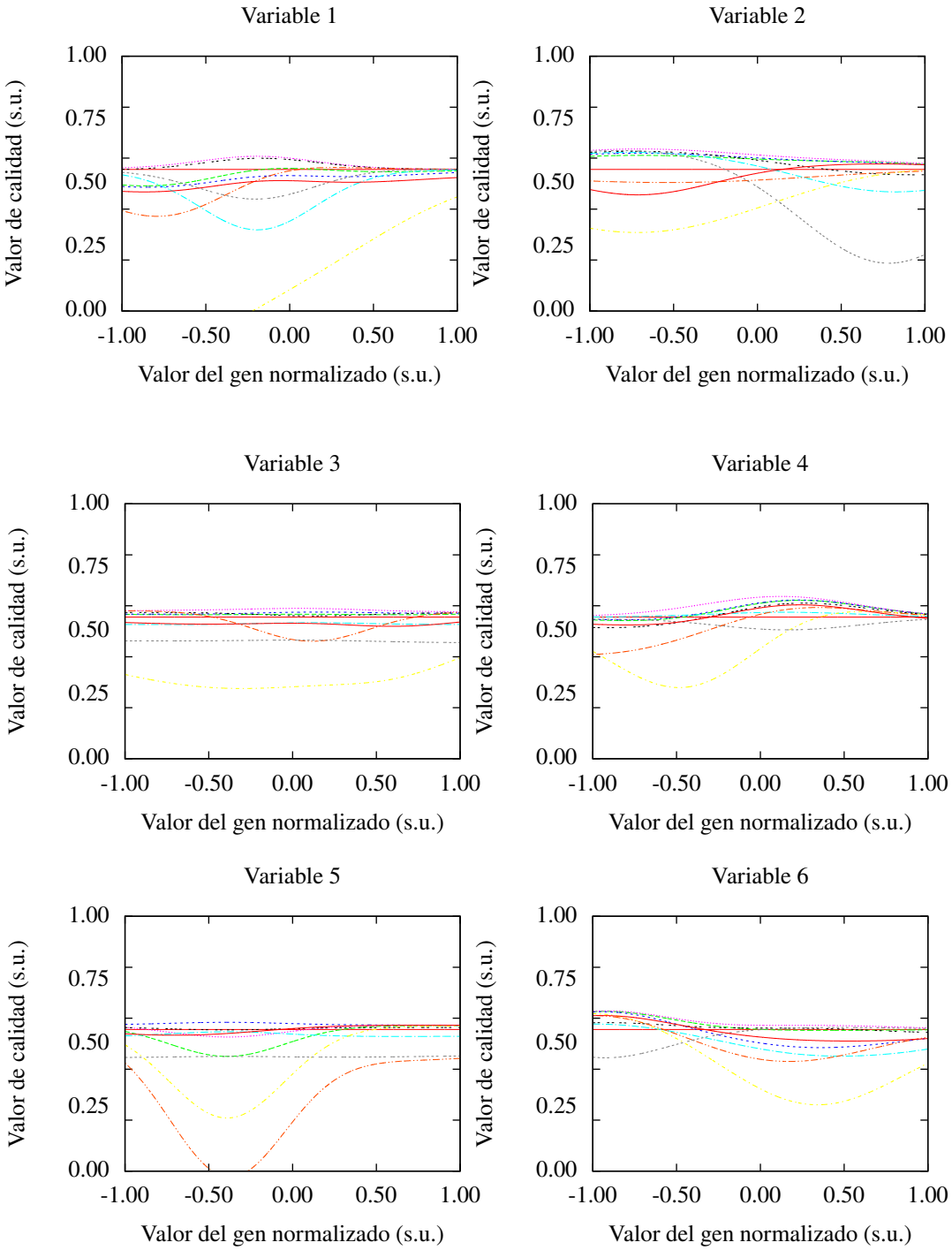
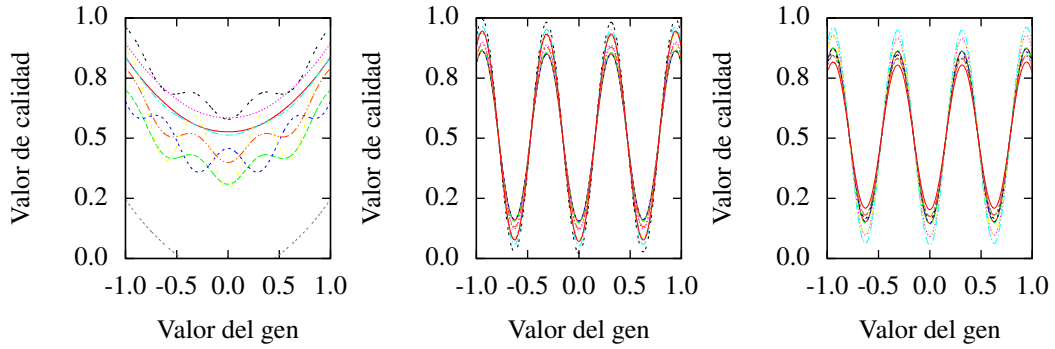


Figura 4.11: Análisis gráfico de separabilidad para la función *Hartman 6*. De izquierda a derecha y de arriba a abajo se representan las 6 variables analizadas. Cada línea de color se corresponde con el análisis de un punto fijo y un barrido aleatorio en la variable indicada en el título. El eje de abscisas representa el valor de las variables normalizado al rango [-1:1] y el eje de ordenadas representa el aporte a la calidad de cada variable según el punto fijo analizado.



(a) Dimensión 10 (b) Dimensión 30 (c) Dimensión 50

Figura 4.12: Análisis gráfico de separabilidad: Función *Griewank*. Se presentan los resultados del análisis de separabilidad para una variable en dimensiones 10, 30 y 50. Como se observa el grado de separabilidad es diferente dependiendo de la dimensión del problema.

et al., 1995], donde los autores aconsejaban utilizar más funciones *No-Separables* en los conjuntos de prueba ya que estas presentaban mayor dificultad para los algoritmos evolutivos.

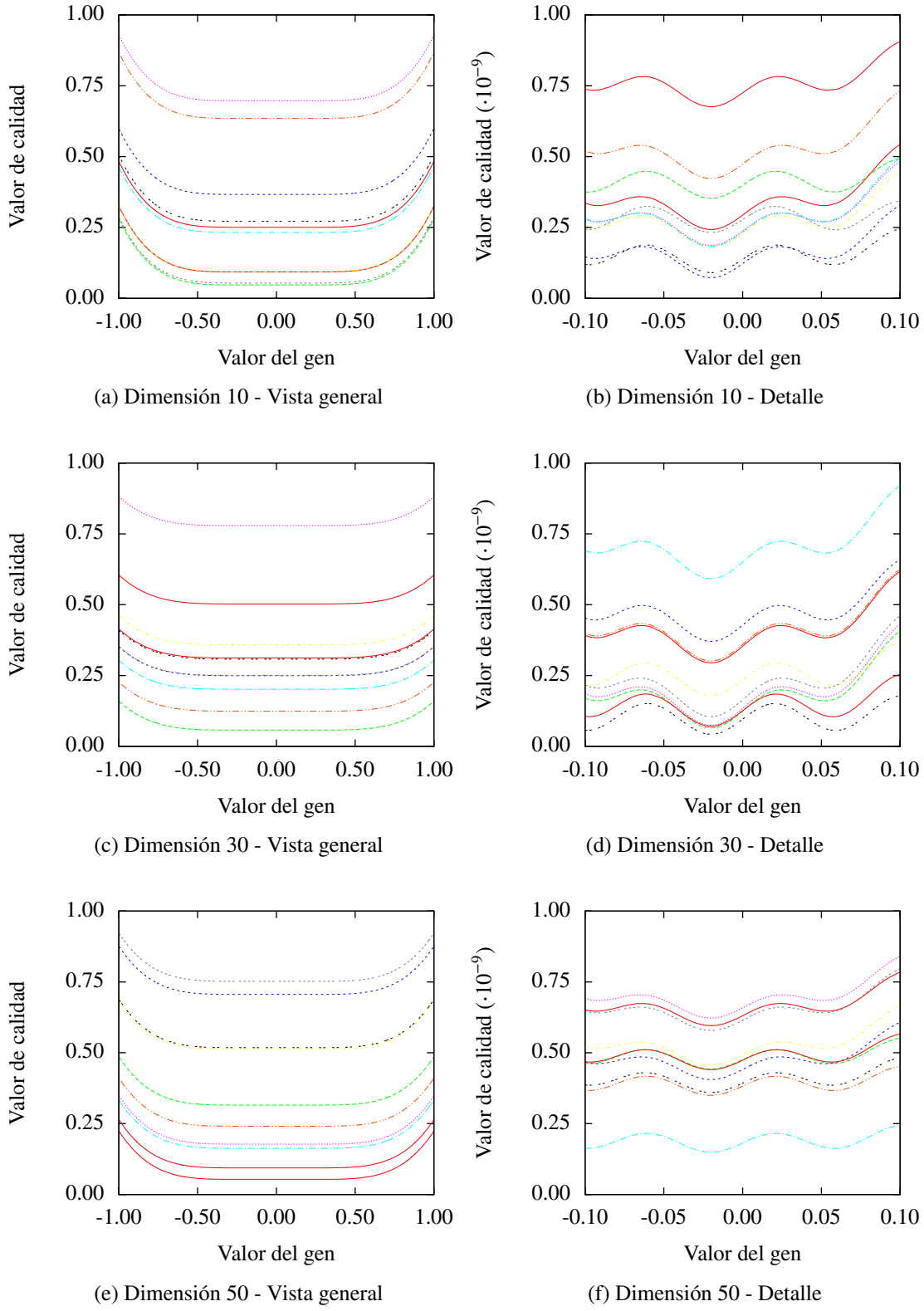
El método aplicado es sencillo y proporciona resultados claros, sin embargo, hay funciones que requieren un análisis más profundo o una vista más detallada de las gráficas generadas. El primero de los casos que requiere un análisis en mayor detalle es la función *Griewank*. Como se ve en la tabla 4.3, dependiendo de la dimensión del problema, la función es *No-separable* (en el caso de dimensión 10) o *NL-Separable* (para dimensiones 30 y 50). En la figura 4.12 se muestran las gráficas obtenidas para una variable (se han analizado todas las variables, pero por cuestión de espacio solamente se muestra una, considerada representativa) tras la aplicación del método explicado en la sección 4.1.3 sobre esta función. Como se puede observar en la gráfica 4.12a, la función es *No-Separable* ya que, además de presentar diferentes ``formas" gráficas para cada punto fijo utilizado, el punto óptimo de cada punto fijo se encuentra en un punto $x$ diferente. Sin embargo, para la misma función utilizando 30 y 50 parámetros (figuras 4.12b y 4.12c) las ``formas" gráficas son diferentes, pero el óptimo se sitúa en el mismo punto fijo para todos los puntos fijos analizados. Por tanto, en estas dimensiones la función es de tipo *NL-Separable*. En [Locatelli, 2003], los autores demuestran como la topología de esta función se suaviza a medida que crecen las dimensiones, afectando a la separabilidad de la función como hemos visto en el análisis de separabilidad realizado.

También es necesario un análisis detallado de las funciones *Penalized 1* y *Penalized 2*. En las figuras 4.13a, 4.13c, 4.13e, 4.14a, 4.14c y 4.14e se muestra el resultado gráfico de la ejecución del algoritmo de análisis sobre estas dos funciones con los parámetros utilizados por defecto. Con el objetivo de resumir este análisis se muestra solamente una variable de las estudiadas. A la vista de estas gráficas, la conclusión del análisis sería determinar que estas dos funciones son del tipo *L-Separables*, ya que todas las gráficas son paralelas entre sí. Sin embargo, realizando un análisis más detallado, como el que se muestra en las figuras 4.13b, 4.13d, 4.13f, 4.14b, 4.14d y 4.14f, la conclusión es diferente. Estas figuras muestran como, dependiendo del punto fijo analizado, la ``forma" de las gráficas es diferente siendo las funciones de tipo *NL-Separables*, ya que el óptimo de todos los puntos fijos se encuentra en la misma coordenada x.

Las funciones *Bohachevsky 1* y *Bohachevsky 2* requieren el mismo análisis que las funciones anteriores. En un primer estudio, ambas funciones parecen iguales y además *L-Separables*

Función	Dimensión	Separabilidad
Aluffi-Pentini's	2	L-Separable
Axis Parallel Hyperellipsoid	10, 30, 50	L-Separable
Becker and Lago	2	L-Separable
Bohachevsky 1	2	L-Separable
Cosine Mixture	10, 30, 50	L-Separable
Rastrigin	10, 30, 50	L-Separable
Schwefel	10, 30, 50	L-Separable
Schwefel 2.22	10, 30, 50	L-Separable
Sphere Model	10, 30, 50	L-Separable
Step	10, 30, 50	L-Separable
Sum of Different Power	10, 30, 50	L-Separable
Ackley	10, 30, 50	NL-Separable
Easom	2	NL-Separable
Griewank	30, 50	NL-Separable
Levy	10, 30, 50	NL-Separable
Penalized 1	10, 30, 50	NL-Separable
Penalized 2	10, 30, 50	NL-Separable
Schwefel 2.21	10, 30, 50	NL-Separable
Beale	2	No-Separable
Bohachevsky 2	2	No-Separable
Colville	4	No-Separable
Dekkers and Aarts	2	No-Separable
Goldstein Price	2	No-Separable
Griewank	10	No-Separable
Hartman 3	3	No-Separable
Hartman 6	6	No-Separable
Kowalik's	4	No-Separable
Matyas	2	No-Separable
Perm	10, 30, 50	No-Separable
Rosenbrock	10, 30, 50	No-Separable
Schwefel 1.2	10, 30, 50	No-Separable
Shekel Family 5	4	No-Separable
Shekel Family 7	4	No-Separable
Shekel Family 10	4	No-Separable
Shekel's Foxholes	2	No-Separable
SixHump Camel Back	2	No-Separable
Zakharov	10, 30, 50	No-Separable

Tabla 4.3: Clasificación de las funciones de prueba en términos de separabilidad.



(a) Dimensión 10 - Vista general

(b) Dimensión 10 - Detalle

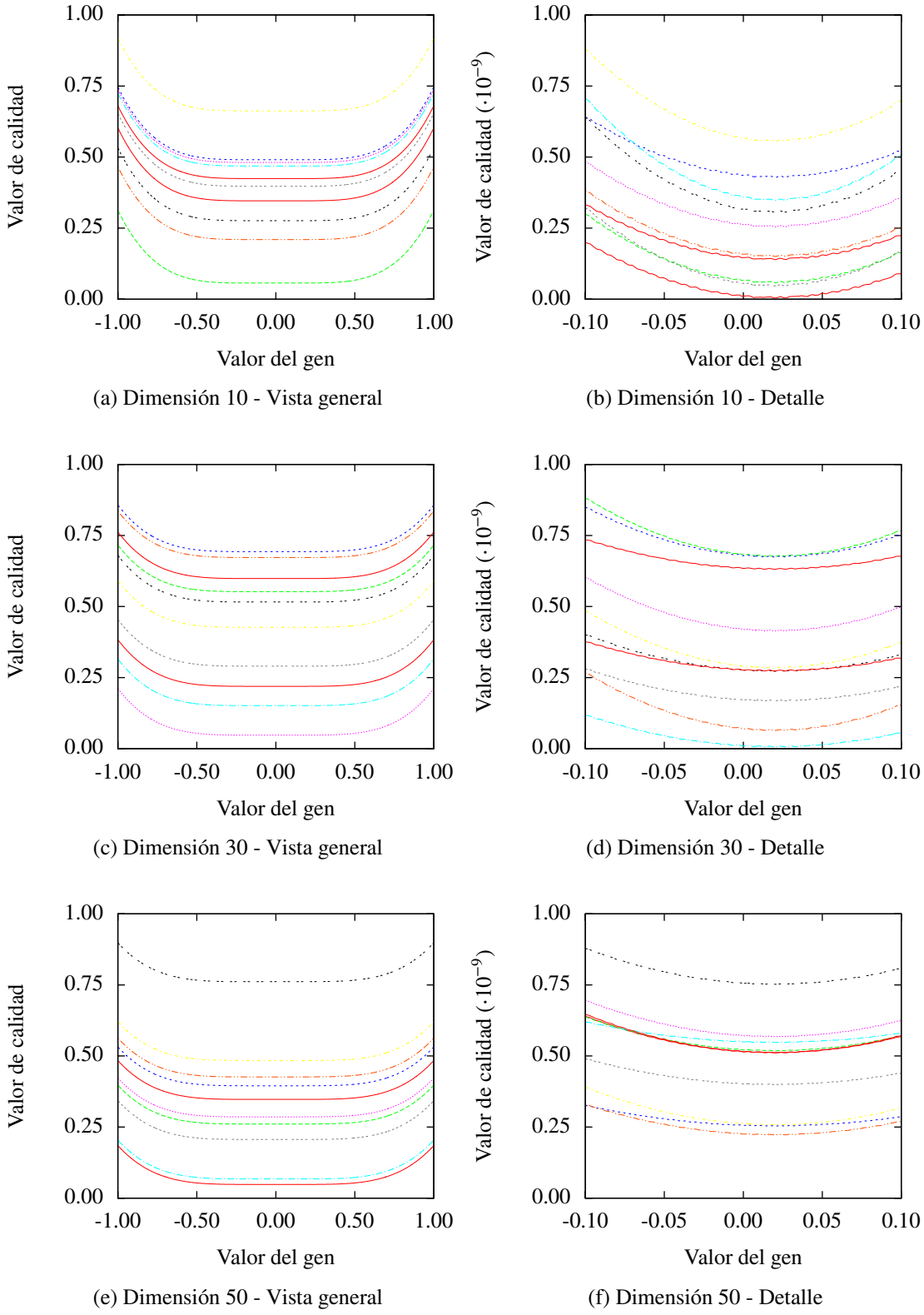
(c) Dimensión 30 - Vista general

(d) Dimensión 30 - Detalle

(e) Dimensión 50 - Vista general

(f) Dimensión 50 - Detalle

Figura 4.13: Análisis gráfico de separabilidad: Función *Penalized 1*. La gráfica de la izquierda muestra el análisis general de la función, analizando en detalle los resultados (gráfica de la derecha) se observa que la función es de tipo *NL-Separable*.



(a) Dimensión 10 - Vista general

(b) Dimensión 10 - Detalle

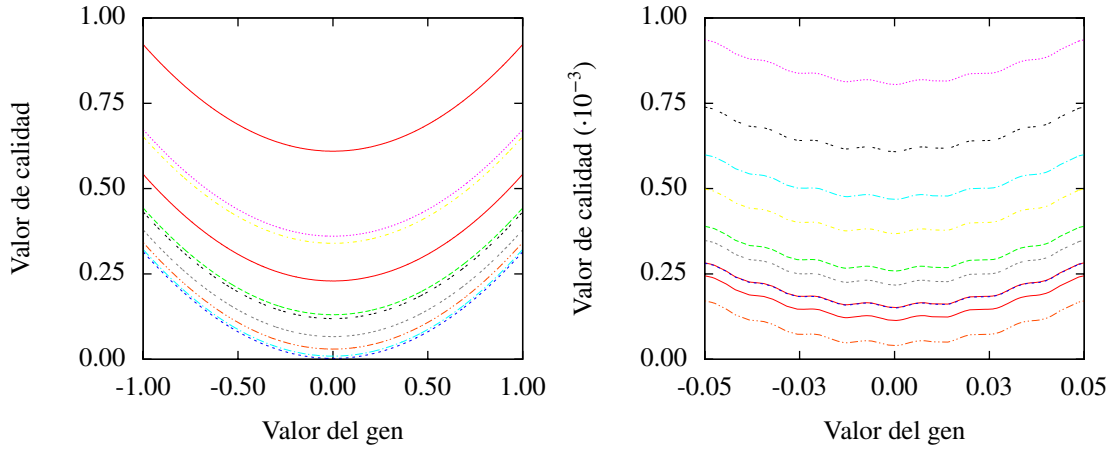
(c) Dimensión 30 - Vista general

(d) Dimensión 30 - Detalle

(e) Dimensión 50 - Vista general

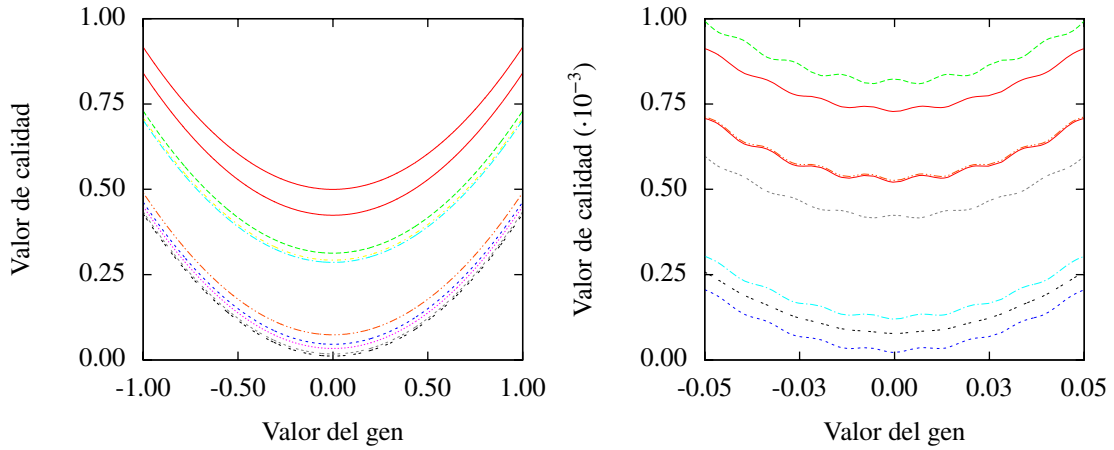
(f) Dimensión 50 - Detalle

Figura 4.14: Análisis gráfico de separabilidad: Función *Penalized 2*. La gráfica de la izquierda muestra el análisis general de la función, analizando en detalle los resultados (gráfica de la derecha) se observa que la función es de tipo *NL-Separable*.



(a) Bohachevsky 1 - Vista general

(b) Bohachevsky 1 - Detalle



(c) Bohachevsky 2 - Vista general

(d) Bohachevsky 2 - Detalle

Figura 4.15: Análisis gráfico de separabilidad: Funciones *Bohachevsky 1* y *Bohachevsky 2*. En cada función, la gráfica de la izquierda muestra el análisis general de la función, analizando en detalle los resultados (gráfica de la derecha).

(ver figuras 4.15a y 4.15c). Sin embargo, los resultados de un análisis más detallado, como el que se muestra en las figuras 4.15b y 4.15d muestran que la función *Bohachevsky 1* es *L-Separable* y la función *Bohachevsky 2* es *No-Separable*.

Como se ha visto en este apartado, la aplicación del algoritmo de separabilidad es sencilla, aunque en ciertas ocasiones sea necesario un análisis más detallado que permita obtener las conclusiones correctas. A partir de los resultados proporcionados por este algoritmo de análisis, es posible clasificar las funciones de forma más detallada y obtener más información acerca de las dependencias entre sus variables que la que conocíamos inicialmente (ver tabla 4.1). Como veremos en la sección de aplicación, esta información nos ayudará a escoger el algoritmo más adecuado para resolverlas. A continuación, se profundizará más en la estimación de la topología de las funciones analizándolas en términos de modalidad.

Modalidad

Notación que será utilizada en esta sección:

- $f : \mathbb{R}^n \longrightarrow \mathbb{R}$, es una función de codificación real.
- $x \in \mathbb{R}^n$, es un vector de parámetros reales.
- x^*, es un óptimo local de la función f.
- $\mathscr{B}(x^*)$, es un centro de atracción del espacio de calidad cuyo óptimo local es x^*.

Definición matemática de modalidad En el campo de la optimización al hablar de modalidad se hace referencia a la existencia o no de óptimos locales y globales en una función. Se distinguen dos tipos de funciones en cuanto a su modalidad: funciones unimodales y funciones multimodales. Las funciones unimodales son aquellas que no presentan óptimos locales y poseen, únicamente, un óptimo global. Por el contrario, las funciones multimodales pueden tener varios óptimos globales y locales.

Como ya se definió en el apartado 4.1.3 de la sección 4.1 de este capítulo, un óptimo local es un punto del espacio de búsqueda que presenta el mejor valor de calidad dentro de una región reducida de dicho espacio. Formalmente, si estamos minimizando la función $f : \mathbb{R}^n \longrightarrow \mathbb{R}$, dada $S \in \mathbb{R}^n$ una partición del espacio de búsqueda, se dice que $x_0 \in S$ es un óptimo local si, y sólo si, $\forall x \in S : f(x_0) \leq f(x)$. Por otro lado, un óptimo global es un punto del espacio de búsqueda que cumple $\forall x \in \mathbb{R}^n : f(x_0) \leq f(x)$. Es decir, es un punto que presenta el mejor valor de calidad de entre todos los puntos pertenecientes al espacio de búsqueda.

Ambos tipos de funciones, tanto unimodales como multimodales, son de interés para el análisis del comportamiento de los AEs. Aunque las funciones unimodales normalmente se resuelven fácilmente mediante algoritmos de búsqueda local, como un descenso de gradiente, desde el punto de vista computacional, es interesante comparar el rendimiento de un AE frente a estos algoritmos de búsqueda local. En el caso de las funciones multimodales, no es recomendable utilizar este tipo de algoritmos, ya que tienen tendencia a quedar atrapados en óptimos locales. Esto también ocurre en el caso de los AEs, y generalmente en cualquier tipo de metaheurística, pero su comportamiento basado en poblaciones hace que la probabilidad de caer en una de las soluciones no óptimas sea menor.

La clasificación de una función como unimodal o multimodal no proporciona información útil a un usuario cuando tiene que elegir qué AE utilizar o cómo configurar un AE para resolver un problema. Dentro de ambos tipos de funciones podemos encontrar diferentes topologías que son más o menos favorables para determinadas estrategias de búsqueda. Por ejemplo, dentro de las funciones unimodales existen las llamadas *aguja en un pajar*, que ya han sido descritas en la sección 4.1.3, y cuya dificultad radica en que el óptimo global de la función está situado en una región del espacio de búsqueda con poca información de gradiente, en muchas ocasiones plana. En el caso de las funciones multimodales el número de óptimos locales y la distribución de los mismos en el espacio de búsqueda son dos características que afectan a las estrategias de búsqueda y que sería deseable poder estimar *a priori* para poder decidir entre uno u otro AE.

Con el objetivo de estimar con mayor detalle la topología de los espacios de calidad, en este trabajo se presentarán dos algoritmos de caracterización de los mismos en términos de modalidad. Antes de explicar el funcionamiento concreto de estos algoritmos se realizará una revisión de la teoría de centros de atracción en la cual se fundamentan.

Teoría de centros de atracción El algoritmo de análisis de espacios de calidad que se presentará en esta sección está basado en la teoría de centros de atracción. Un centro de atracción de un óptimo local $\mathscr{B}(x^*)$ se define como un subconjunto de puntos del espacio de búsqueda tales que, comenzando una búsqueda local en cada uno de esos puntos, en un número de paso finitos, el proceso de búsqueda termina en dicho óptimo local. Esto permite dividir el espacios de calidad en particiones, cada una de ellas representa un centro de atracción [Garnier and Kallel, 2000]. Gráficamente, suponiendo un espacio de búsqueda bi-dimensional con cuatro mínimos locales s_1,..., s_4 (como se ve en la Figura 4.16) cada óptimo local tiene asociado su cuenca de atracción $\mathscr{B}_1$,..., $\mathscr{B}_4$.

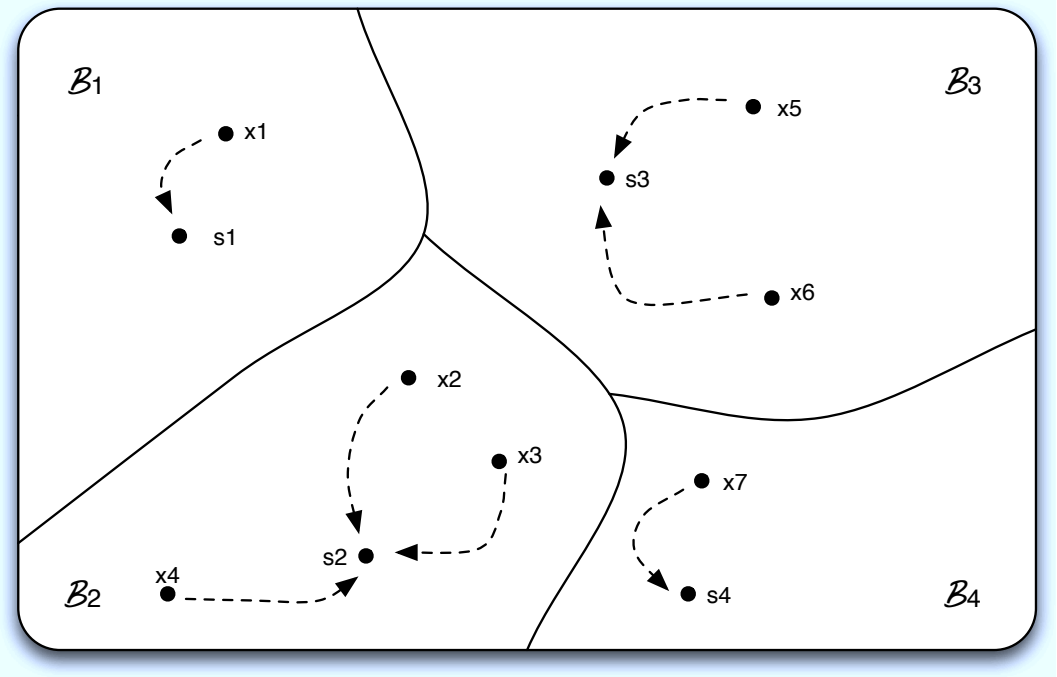


Figura 4.16: Representación gráfica de un espacio de calidad con cuatro mínimos. Este espacio de búsqueda puede ser dividido en 4 particiones, cada una de ellas se corresponde con una cuenca de atracción de un óptimo local (s_1,..., s_4).

La distribución de los óptimos locales y el tamaño de las cuencas de atracción de dichos óptimos es un factor importante para el rendimiento de las metaheurísticas y, por lo tanto, para los AEs. La dificultad de optimización de un problema está ligada a la existencia o no de óptimos locales, el número de ellos, la densidad espacial, el tamaño de sus cuencas de atracción, etc.

En [Garnier and Kallel, 1999, Garnier and Kallel, 2002] los autores proponen una metodología para estimar el número de óptimos locales de un problema y el tamaño de los centros de atracción de cada uno de esos óptimos. El método que proponen consiste en realizar un proceso de búsqueda local a partir de N puntos aleatorios del espacio de búsqueda, suponiendo que cada búsqueda local converge en un único óptimo. Basándose en los resultados obtenidos tras las N búsquedas, el método propuesto permite obtener β_j ($0 \leq j < N$), que se corresponde con el número de óptimos locales detectados a partir de j puntos. Con esta información, los autores obtienen una familia de funciones de densidad de probabilidad H_γ ($\gamma \in \mathbb{R}$) que representan la probabilidad de obtener centros de atracción de tamaño γ. A partir de esas funciones de probabilidad es posible estimar el número medio de óptimos locales encontrados a partir de j soluciones aleatorias ($\beta_{j,\gamma}$). Para estimar la distribución más pro-

bable de H_γ utilizan un test χ^2 y de esta forma deducen el número de óptimos locales y la distribución del tamaño de los centros de atracción de un problema determinado.

Además del estudio teórico del problema, los autores validaron la metodología propuesta en problemas de codificación binaria obteniendo resultados satisfactorios que verificaron la bondad de la metodología. Para ampliar los resultados obtenidos en estos trabajos, otros autores utilizaron esta metodología para analizar los espacios de calidad de otros problemas conocidos. En [Ochoa et al., 2008] los autores analizan versiones combinatorias de espacios de calidad *NK* generando redes de óptimos locales que ayudan a estimar la dificultad de los problemas. En [Albrecht et al., 2010, Albrecht et al., 2008] se analizan problemas *k-SAT*, problemas de satisfabilidad, y se obtienen las distribuciones de probabilidad de óptimos locales en instancias con pocas variables.

A pesar de los buenos resultados de estos trabajos, su ámbito de aplicabilidad se reduce a problemas de codificación binaria o a problemas de optimización combinatoria y, por lo tanto, sus resultados no son generalizables a problemas de codificación real. Por este motivo, tomando como base dichos trabajos, en este trabajo se ha desarrollado un nuevo algoritmo de caracterización de espacios de calidad de codificación real. Este algoritmo permitirá analizar los espacios de calidad estimando sus topología teniendo en cuenta sus óptimos locales, los tamaños de sus centros de atracción y la distribución de los mismos.

Algoritmos de caracterización de espacios de calidad en términos de modalidad El algoritmo que se presenta en esta sección permite estudiar la topología del espacio de calidad en términos de distribución y tamaño de los centros de atracción. En consecuencia, la primera información que proporciona es la clasificación de las funciones en unimodales frente a multimodales. El algoritmo es una variación del método presentado en [Garnier and Kallel, 1999, Garnier and Kallel, 2002] para problemas de codificación real. Como en el caso del algoritmo original, la entrada es una muestra de puntos del espacio de búsqueda de tamaño N, $\Omega = \{x_1, \ldots, x_N\}$, $x_i \in \mathbb{R}^n$. Para cada punto de dicho conjunto se aplica un algoritmo de búsqueda local. En la implementación utilizada en este trabajo, este algoritmo consiste en generar p vecinos aleatorios aplicando un operador de modificación μ. Todos los μ-vecinos son evaluados, se elige el mejor de ellos y la búsqueda continua desde ese punto, es decir, desde el vecino con mejor valor de calidad. El punto x_i se modifica aplicando μ hasta que no se encuentre un vecino mejor en el vecindario de x_i. Llegados a este punto se considera que x_i es un óptimo local.

La salida que proporciona el algoritmo es una lista de óptimos locales y su frecuencia de aparición, es decir, cuántos de esos N puntos aleatorios finalizaron su búsqueda local en ese óptimo local. Con esta información, el cálculo del tamaño del centro de atracción se realiza como $\alpha_j = k/N$, donde k es el número de puntos en Ω que han alcanzado ese óptimo local.

En el algoritmo 1 se muestra el pseudocódigo que implementa el procedimiento de análisis de espacios de calidad. Este procedimiento se aplica a $N = |\Omega|$ puntos aleatorios del espacio de búsqueda. El operador μ genera $|R|$ puntos aleatorios dentro de un área de radio r a partir del punto x^*. Tras la ejecución del proceso de búsqueda cada uno de los puntos iniciales x_i habrá alcanzado un óptimo local $m_j \in E$. Para cada uno de esos óptimos locales, se deben contar el número de puntos aleatorios que los ha alcanzado, es decir cuántos puntos de Ω pertenecen a su centro de atracción $\mathcal{B}(m_j)$. Como se trabaja con parámetros de codificación real, se establece un valor umbral δ. Así, supondremos que una solución del espacio de búsqueda pertenece a un centro de atracción si la distancia euclídea entre la solución encontrada y el óptimo local del centro de atracción es menor que δ. Resumiendo, el algoritmo

Algoritmo 1 Algoritmo de análisis de modalidad

Entrada: Una función de calidad $f : \mathbb{R}^n \rightarrow \mathbb{R}$, una lista de puntos $\Omega = \{x_1, \ldots, x_n\}$ y un operador de búsqueda local μ.

Salida: Una estimación del número de óptimos locales de la funció y el tamaño de cada uno de ellos.

```
for all x_i ∈ Ω do
  x* = x_i
  while f(x*) ≤ f(x_i) do
    Generar R como R = {x_j | x_j = μ(x*)}
    Seleccionar el mejor elemento de R.
  end while
  // E es el conjunto de óptimos locales encontrado por el algortimo, inicialmente está vacio.
  for all m_j ∈ E do
    if distancia(x*, m_j) ≤ δ then
      Asignar x* a B(m_j)
    else
      Crear un nuevo centro de atracción B(x*)
    end if
  end for
end for
```

presentado permite:

- Obtener el número de óptimos locales de un espacio de calidad y, como consecuencia, detectar si la función es unimodal o multimodal.
- Estimar la topología de un espacio de calidad, la cual está directamente relacionada con su dificultad. En particular, la información con mayor relevancia es la siguiente:
  - El tamaño de los centros de atracción y, más concretamente, el tamaño del centro de atracción óptimo y el más amplio.
  - La distribución de los centros de atracción, distancia entre los mismos, etc.

En el caso de que la función sea unimodal, para estimar su particularidades, es necesaria más información además de conocer que posee únicamente un óptimo global. En las funciones unimodales existe siempre un camino hacia el óptimo. Las principales dificultades para determinar y seguir dicho camino están relacionadas con su longitud y con la monotonía de la superficie. Estas dos caracteríticas están relacionadas y las funciones que las presentan se conocen típicamente como problemas *longpath* [Horn et al., 1994] y problemas *needle-in-a-haystack* [Goldberg, 1989b], respectivamente. Los problemas con caminos cortos hasta el óptimo son más fáciles de resolver que aquellos que presentan caminos de mayor longitud, los cuales son un reto para los algoritmos de optimización.

Para obtener más información acerca de las funciones unimodales, se ha desarrollado un método de búsqueda local que mide la longitud del camino más largo hasta el óptimo. Los puntos iniciales de la búsqueda local son las esquinas del espacio de búsqueda, es decir, los 2^n puntos $x_i = (x_i^1, \ldots, x_i^n)$, con $x_i^j \in \{-1.0, 1.0\}$. x^* es el óptimo global de la función f y μ

Algoritmo 2 Algoritmo para calcular la longitud del camino al óptimo.

Entrada: Una función de calidad $f : \mathbb{R}^n \rightarrow \mathbb{R}$, una lista de puntos $X = \{x_1, \ldots, x_n\}$ y un operador de búsqueda local μ.

Salida: La longitud del camino más largo hasta el óptimo.

for all $x_i \in X$ **do**
  for $j = 1 \ldots n$ **do**
    while $f(x_i) > f(x^*)$ **do**
      Modificar x_i aplicando el operador μ.
      if $f(\mu(x_i)) \leq f(x_i)$ **then**
        $x_i = \mu(x_i)$
        $longitud_{ij} = longitud_{ij} + 1$
      end if
    end while
    $longitud_i = max(longitud_i, longitud_{ij})$
  end for
end for
Devolver $longitud = max\{longitud_i; i \in [0, |X|]\}$

es un operador de modificación que opera de la siguiente forma:

$$\mu(x_t^i) = x_i^{t-1} + (x_i^{t-2} - x_i^{t-1})/rand(0, 100) \tag{4.12}$$

De esta forma, la longitud del camino puede ser medida aplicando un método de descenso de gradiente. El algoritmo 2 muestra el pseudocódigo del procedimiento aplicado.

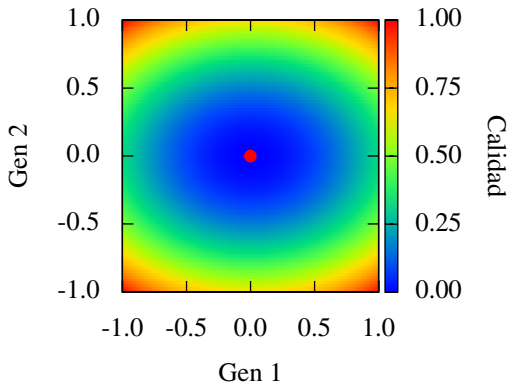
Ejemplos de aplicación del algoritmo de análisis de modalidad A continuación se muestra el funcionamiento de los algoritmos de modalidad presentados tras su aplicación a tres funciones: *Axis Parallel Hyper Ellipsoid*, *Cosine Mixture* y *Schwefel*. Las expresiones analíticas de estas funciones son las siguientes:

$$f_{Axis}(x) = \sum_{i=1}^{n} ix^2 \tag{4.13}$$

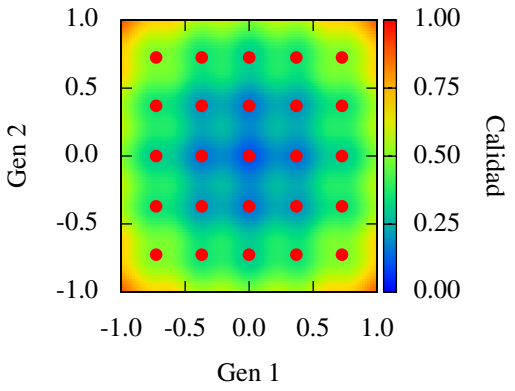
$$f_{Cosine}(x) = 0.1 \sum_{i=1}^{n} cos(5.0\pi x_i) - \sum_{i=1}^{n} x_i^2 \tag{4.14}$$

$$f_{Schwefel}(x) = -\sum_{i=1}^{n} x_i sin(\sqrt{|x_i|}) \tag{4.15}$$

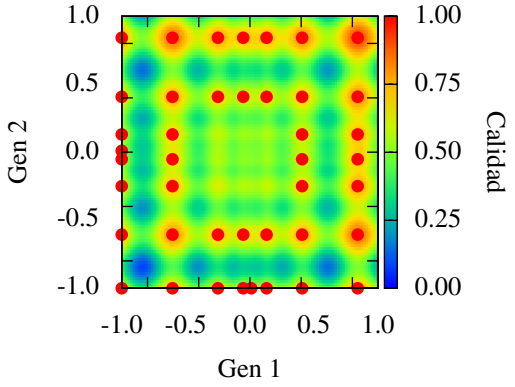
Estas funciones son escalables en cuanto a dimensionalidad. Para demostrar la corrección del método presentado se muestran ejemplos utilizando dos dimensiones. El primer paso del análisis de modalidad consiste en ejecutar el algoritmo 1 sobre las funciones escogidas para tratar de localizar los centros de atracción de las mismas. En esta primera prueba, el tamaño de la muestra de puntos aleatorios se estableció en un valor fijo de 1000 puntos. Los resultados obtenidos para las tres funciones utilizadas se muestran en la figura 4.17. En dichas figuras se representan las curvas de nivel de las funciones, indicando el valor de calidad de cada zona, y los centros de atracción detectados por el algoritmo, indicados por un punto rojo.



(a) Función Axis Parallel



(b) Función Cosine Mixture



(c) Función Schwefel

Figura 4.17: Resultados de aplicación del algoritmo de modalidad en funciones de dos dimensiones: funciones *Axis Parallel*, *Cosine* y *Schwefel*.

Analizando los resultados obtenidos se distinguen dos tipos de funciones: una función en la que sólo se ha detectado un centro de atracción (figura 4.17a) y dos funciones en las que se han detectado varios centros de atracción (figuras 4.17b y 4.17c). La función *Axis Parallel Hyper Ellipsoid* es una función unimodal, ya que solamente se ha detectado un centro de atracción. Por ser una función unimodal cumple la propiedad de que desde cualquier punto del espacio de búsqueda existe un camino que en un número finito de pasos encuentra el óptimo de dicha función. Como ya se ha mencionado en apartados anteriores, en este trabajo la dificultad de una función unimodal se medirá con respecto a la longitud del camino más largo hasta el óptimo. Para obtener esta longitud, sobre esta función se ha aplicado el algoritmo 2 obteniendo como resultado una longitud de camino de 147 pasos. En la figura 4.18 se muestran gráficamente los caminos hacia el óptimo analizados en esta función.

Las funciones *Cosine Mixture* y *Schwefel*, al contrario que la función *Axis Parallel Hyper Ellipsoid*, presentan más de un centro de atracción. Por lo tanto, son funciones multimodales. Para analizar este tipo de funciones se debe estudiar la distribución de los centros de atracción en cuanto a tamaño y distancia al óptimo. En la figura 4.19 se muestran dos gráficas en las

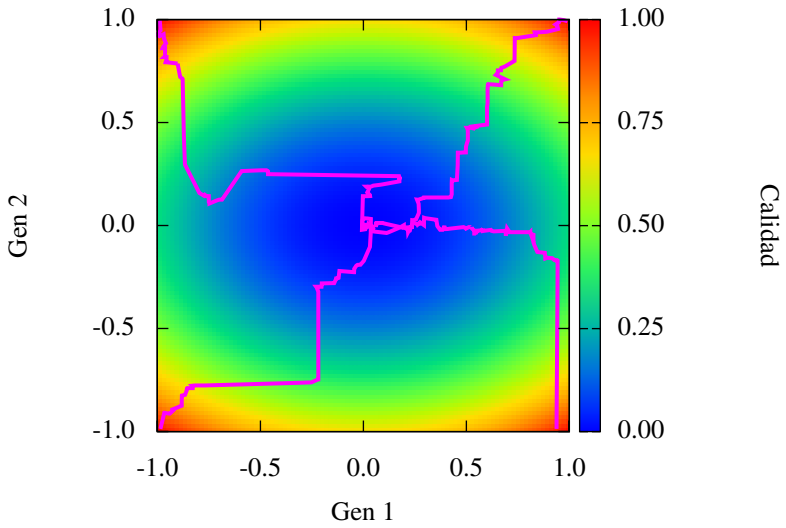


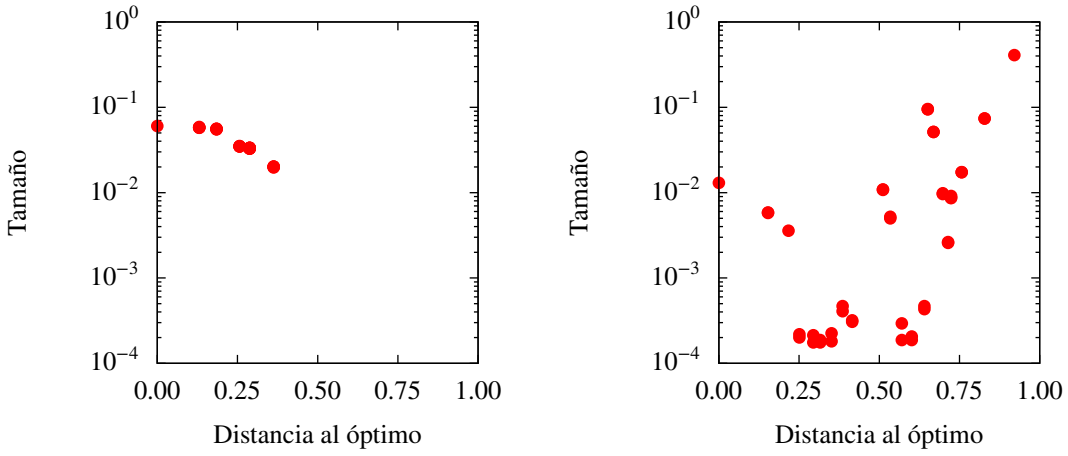
Figura 4.18: Caminos hacia el óptimo de la función *Axis Parallel Hyper Ellipsoid*

que se presenta la distribución de centros de atracción de estas dos funciones. En el eje de ordenadas se representa la distancia al óptimo de la función y en el eje de abscisas, utilizando una escala logarítmica, el tamaño de centro de atracción medido como el porcentaje de puntos iniciales que terminan su búsqueda en ese óptimo local.

Como se puede observar en las gráficas de distribución, la topología de las funciones *Cosine Mixture* y *Schwefel* es diferente. Si nos fijamos en la distancia a los centros de atracción (eje de ordenadas), en la primera función (ver figura 4.19a), la distancia entre centros de atracción es corta. Sin embargo, en el caso de la función *Schwefel*, los centros de atracción están separados por grandes distancias, siendo la distancia máxima entre los mismos de 0.92 unidades, cuando la distancia máxima posible es de 1.0 unidades. Teniendo en cuenta la otra característica, el tamaño de los centros de atracción (eje de abcisas), en la primera función dicho tamaño decrece a medida que nos alejamos del óptimo. En la función *Schwefel* ocurre el caso contrario, el tamaño de los centros de atracción crece con la distancia al óptimo.

Como se ha demostrado en esta sección, con la información que proporcionan los dos algoritmos de análisis de modalidad el usuario puede, en primer lugar, identificar si una función es unimodal o multimodal. Una vez determinado el tipo de función con la que trata y en el caso de ser multimodal, el algoritmo le permite estimar la distribución de los centros de atracción y, con esta información, obtener propiedades de la topología del espacio de calidad. En caso contrario, si la función fuese unimodal, tras la aplicación del algoritmo 2, el usuario puede estimar la dificultad de la función a partir de la información de longitud del camino al óptimo.

Resultados del análisis de modalidad El conjunto de funciones de prueba utilizado en este trabajo incluye diferentes tipos de funciones en cuanto a su modalidad (veáse apéndice B). En este conjunto existen algunas funciones de dos dimensiones que pueden ser analizadas utilizando su representación gráfica. Sin embargo, la gran mayoría son funciones de más de dos dimensiones e incluso funciones escalables, en las cuales no es posible utilizar una representación gráfica de su superficie para analizarlas. Los algoritmos presentados en esta sección permiten realizar una estimación de la topología de espacios de calidad de más de



(a) Función Cosine Mixture (b) Función Schwefel

Figura 4.19: Distribución de centros de atracción en las funciones *Cosine Mixture* y *Schwefel* de dos dimensiones. El eje de abscisas representa la distancia al centro de atracción óptimo. El eje de ordenadas representa el tamaño del centro de atracción.

dos dimensiones.

En primer lugar, se aplicó el algoritmo 1 sobre el conjunto de funciones para determinar si eran unimodales o multimodales. En el caso de las funciones multimodales se analizó su topología teniendo en cuenta los centros de atracción encontrados. La información obtenida se muestra en la tabla 4.4, la cual incluye el número de centros de atracción, el tamaño del centro de atracción más amplio, el tamaño del centro de atracción que contiene a la solución óptima, la distancia entre ambos centros de atracción y la distancia máxima entre centros de atracción.

Para analizar las funciones multimodales en mayor detalle, en las figuras 4.20 y 4.21 se muestran las gráficas de distribución de centros de atracción. Como en el apartado anterior, estas gráficas muestran en el eje de ordenadas la distancia al centro de atracción óptimo y en el eje de abscisas el tamaño del centro de atracción. A partir de estas gráficas y los datos de la tabla 4.4 se puede estimar la topología de un espacio de calidad multimodal.

Si tras la aplicación del primer algoritmo no se detecta más de un centro de atracción, la función se clasifica como unimodal. Para estimar su dificultad se aplicó el algoritmo 2 sobre el conjunto de funciones unimodales obteniendo los resultados que se muestran en la tabla 4.5. En esta tabla el dato que se muestra se corresponde con la longitud media del camino al óptimo. A mayor longitud, mayor dificultad para los algoritmos.

El contenido de este apartado se ha centrado en la primera etapa del procedimiento de caracterización. Para ello se ha explicado qué es un espacio de calidad y por qué motivo es importante para los AEs. A partir de una revisión bibliográfica se han determinado qué propiedades de estos espacios son más relevantes para el comportamiento y rendimiento de un AE. Para el análisis de estas características topológicas se han desarrollado tres algoritmos que permiten estimar dichas propiedades de los espacios de calidad. Estos algoritmos han sido ejecutados sobre un conjunto de funciones de prueba muy representativo, que incluye

Función	m	Tamaño del c.a.[a]		Distancia[d]	
		Mayor[b]	Óptimo[c]	May.-Opt.[e]	Máxima[f]
Aluffi	10	$8.60E{-}1$	$8.60E{-}1$	$0.00E{+}0$	$7.00E{-}2$
Becker	4	$1.00E{+}0$	$1.00E{+}0$	$0.00E{+}0$	$0.00E{+}0$
Bohac1	13	$7.54E{-}1$	$7.54E{-}1$	$0.00E{+}0$	$9.00E{-}3$
Cosine	$\approx 5^n$	$2.60E{-}2$	$2.60E{-}2$	$0.00E{+}0$	$2.15E{-}1$
Rastrigin	$\approx 11^n$	$8.00E{-}3$	$8.00E{-}3$	$0.00E{+}0$	$4.86E{-}1$
Schwefel	$\approx 8^n$	$2.36E{-}1$	$2.96E{-}5$	$9.20E{-}1$	$9.20E{-}1$
Ackley's	$\approx 65^n$	$1.00E{-}3$	$1.00E{-}3$	$0.00E{+}0$	$5.00E{-}1$
Griewank	$\approx 4^n$	$1.00E{-}3$	$2.00E{-}4$	$3.97E{-}1$	$4.55E{-}1$
Levy	$\approx 5^n$	$5.77E{-}1$	$5.77E{-}1$	$0.00E{+}0$	$2.61E{-}1$
Penalized1	$\approx 5^n$	$1.28E{-}1$	$1.28E{-}1$	$0.00E{+}0$	$1.13E{-}1$
Penalized2	$\approx 5^n$	$1.74E{-}1$	$1.74E{-}1$	$0.00E{+}0$	$5.00E{-}2$
Beale	4	$4.69E{-}1$	$4.69E{-}1$	$0.00E{+}0$	$5.91E{-}1$
Bohac2	10	$5.35E{-}1$	$5.35E{-}1$	$0.00E{+}0$	$9.00E{-}3$
Deckers	3	$8.79E{-}1$	$8.79E{-}1$	$0.00E{+}0$	$2.91E{-}1$
Goldstein	6	$5.03E{-}1$	$5.03E{-}1$	$0.00E{+}0$	$4.24E{-}1$
Hartman3	3	$6.22E{-}1$	$6.22E{-}1$	$0.00E{+}0$	$4.66E{-}1$
Hartman6	2	$6.25E{-}1$	$2.74E{-}1$	$4.50E{-}1$	$4.50E{-}1$
Rosenbrock	9	$3.76E{-}1$	$3.76E{-}1$	$0.00E{+}0$	$2.20E{-}1$
Kowaliks	65	$1.28E{-}1$	$2.87E{-}4$	$3.16E{-}1$	$4.27E{-}1$
Shekel5	5	$3.12E{-}1$	$2.61E{-}1$	$2.02E{-}1$	$3.89E{-}1$
Shekel7	7	$2.95E{-}1$	$2.95E{-}1$	$0.00E{+}0$	$3.77E{-}1$
Shekel10	10	$2.40E{-}1$	$2.40E{-}1$	$0.00E{+}0$	$3.81E{-}1$
SixHump	6	$2.65E{-}1$	$2.32E{-}1$	$1.14E{-}1$	$1.40E{-}1$
Foxholes	25	$6.70E{-}2$	$6.70E{-}2$	$0.00E{+}0$	$4.87E{-}1$

[a] Tamaño del centro de atracción medido como el tanto por uno de puntos del espacio de calidad que pertenecen a dicho centro.

[b] Tamaño del centro de atracción más amplio del espacio de calidad.

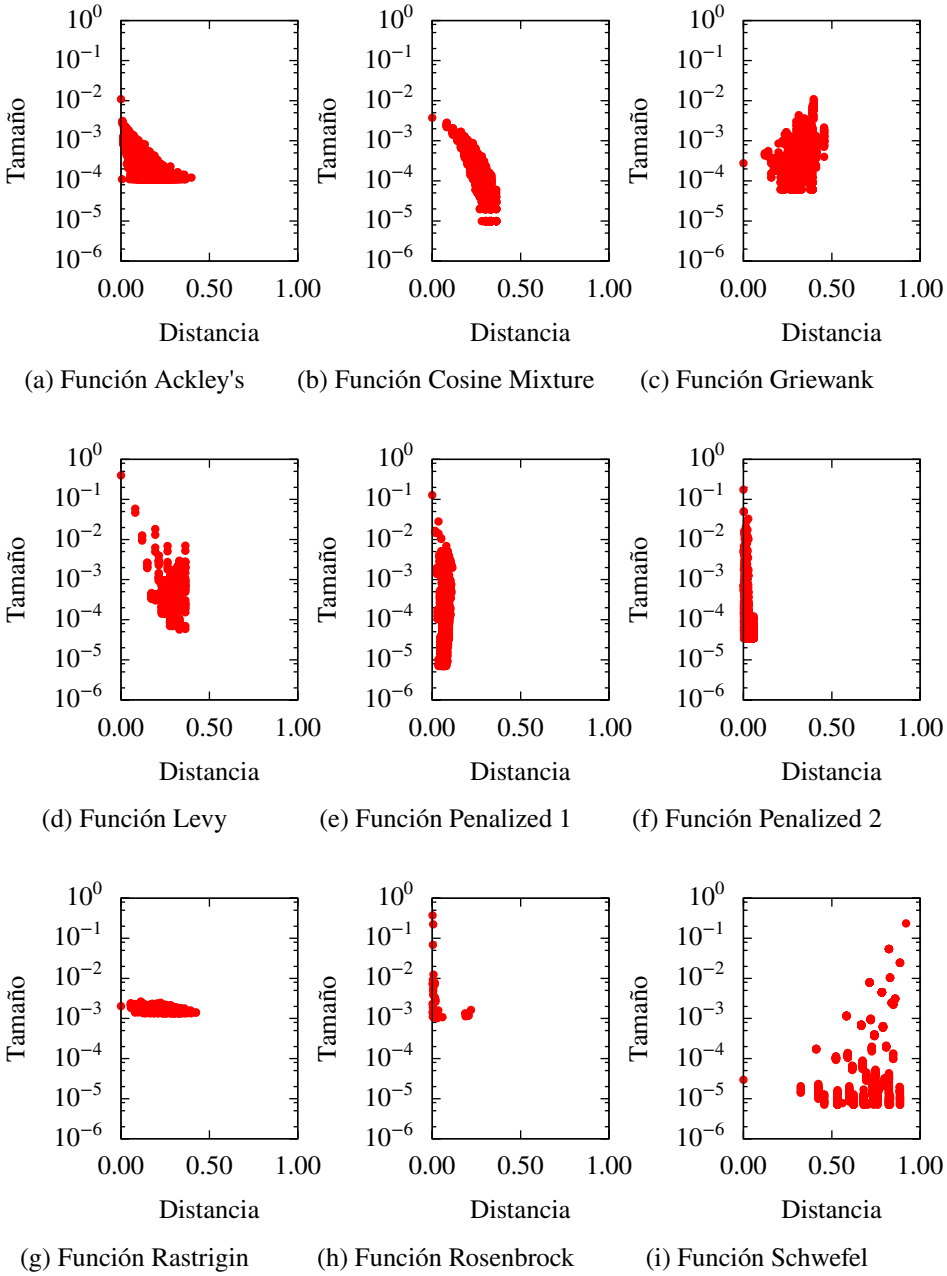
[c] Tamaño del centro de atracción óptimo del espacio de calidad.

[d] Distancia entre centros de atracción.

[e] Distancia entre el centro de atracción más amplio y el óptimo. Este valor es representativo cuando estos centros de atracción son diferentes.

[f] Distancia máxima entre centros de atracción. Es relevante como medida para conocer como de separados se encuentran los centros de atracción del espacio de calidad.

Tabla 4.4: Análisis de modalidad: centros de atracción en funciones multimodales

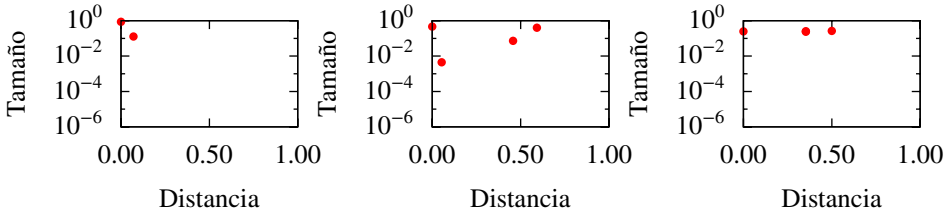


(a) Función Ackley's (b) Función Cosine Mixture (c) Función Griewank

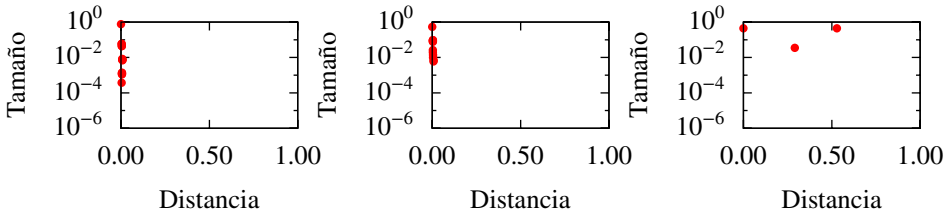
(d) Función Levy (e) Función Penalized 1 (f) Función Penalized 2

(g) Función Rastrigin (h) Función Rosenbrock (i) Función Schwefel

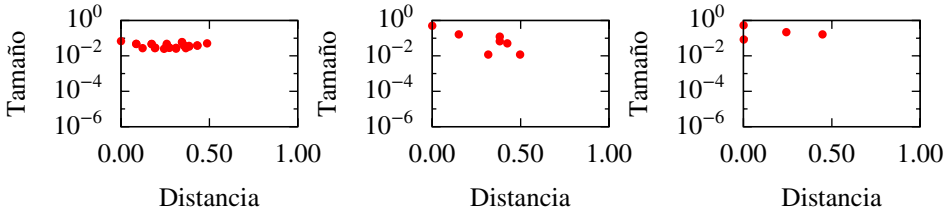
Figura 4.20: Gráficas de distribución de los centros de atracción en el conjunto de funciones escalables. El eje x representa la distancia normalizada hasta el centro de atracción óptimo. El eje y representa el tamaño de los centro de atracción.



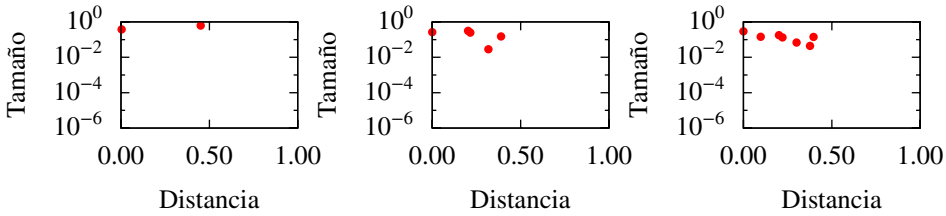
(a) Función Aluffi (b) Función Beale (c) Función Becker



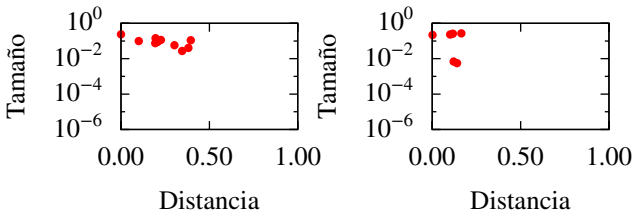
(d) Función Bohachevsky 1 (e) Función Bohachevsky 2 (f) Función Deckers



(g) Función Shekel's Foxholes (h) Función Goldstein (i) Función Hartman 3



(j) Función Hartman 6 (k) Función Shekel 5 (l) Función Shekel 7



(m) Función Shekel 10 (n) Función SixHump

Figura 4.21: Gráficas de distribución de los centros de atracción en el conjunto de funciones Dixon-Szegö. El eje x representa la distancia normalizada hasta el centro de atracción óptimo. El eje y representa el tamaño de los centro de atracción.

Función	Longitud media del camino
Perm	1060.46
Schwefel 2.21	573.48
Colville	508.30
Schwefel 1.2	453.02
Schwefel 2.22	265.15
Axis	224.97
Zakharov	220.41
Sphere	204.12
Matyas	188.31
SumOf	142.14
Step	126.60
Easom	24.40

Tabla 4.5: Análisis de modalidad: longitud del camino al óptimo en funciones unimodales.

funciones con muy diferentes topologías que servirán para caracterizar el comportamiento de los AEs en términos de separabilidad y modalidad. Es importante destacar que esta primera etapa del procedimiento de caracterización debe finalizar siempre con un conjunto de funciones de prueba representativo y caracterizado en base a las propiedades del espacio de calidad que se deseen estudiar. Los diseñadores de AEs pueden proponer sus propios conjuntos de funciones de prueba caracterizados o utilizar los desarrollados en este trabajo.

4.2 Medidas de error y rendimiento para Algoritmos Evolutivos

La segunda de las etapas del procedimiento de caracterización se centra en el establecimiento de las medidas que se utilizarán para analizar los resultados. Siguiendo los criterios aconsejados por [Moreno-Pérez et al., 2007], cada algoritmo debe estudiarse teniendo en cuenta qué propiedades se buscan para la estrategia de búsqueda del mismo. En esta sección se explican diferentes medidas de error y rendimiento utilizadas en otros trabajos del campo de la optimización, detallando sus ventajas e inconvenientes.

Cuando se trabaja con algoritmos de optimización, típicamente, el error de una ejecución se calcula comparándolo con la mejor solución conocida para dicho problema. En cuanto a rendimiento, en el caso de este tipo de algoritmos, esta medida está relacionada con la velocidad de convergencia que, generalmente, se mide con respecto al número de llamadas realizadas a la función de evaluación. Las medidas más utilizadas son el error absoluto, la tasa de éxito y la tasa de rendimiento [Suganthan et al., 2005]. Estas medidas únicamente proporcionan información parcial y, para entender mejor el comportamiento de los algoritmos, es necesario utilizarlas de forma conjunta y, en algunas ocasiones, utilizar medidas complementarias. Como consecuencia, para analizar el comportamiento de un solo algoritmo se generan gran cantidad de datos cuya información está relacionada y es necesario realizar un análisis cruzado de cada una de las medidas para obtener las conclusiones finales.

Para simplificar el proceso de análisis, en este trabajo se propone una nueva medida que combina, en un único valor, resultados a nivel de error y a nivel de rendimiento de los al-

goritmos. Esta medida ha recibido el nombre de medida combinada de error y rendimiento (CPEM) y será utilizada como base para analizar los resultados de los experimentos realizados con los diferentes algoritmos. Se proponen también otras medidas para complementar las capacidades de análisis del CPEM y para comprender mejor el comportamiento de los algoritmos en algunos casos particulares, por ejemplo, el error genotípico. A continuación se detallan las medidas de error y rendimiento propuestas en este procedimiento de caracterización.

Error absoluto

El error absoluto es una de las medidas de error típicas a la hora de comparar algoritmos de optimización. Esta medida proporciona una indicación de la diferencia entre la solución objetivo, x^*, y la solución obtenida por el algoritmo, x. En el caso de los problemas de minimización que se tratarán es este trabajo, cuánto menor sea el valor de esta medida, mejor será el comportamiento del algoritmo en cuestión. El error absoluto se calcula según la siguiente ecuación:

$$error\_absoluto = |f(x) - f(x^*)| \tag{4.16}$$

Es una medida simple y proporciona información útil acerca del algoritmo, pero una de sus desventajas es que no proporciona información acerca del rendimiento de los algoritmos a la hora de obtener las soluciones, es decir, cuántas generaciones o llamadas a la función de calidad necesita un algoritmo para alcanzar ese error absoluto.

Tasa de éxito (SR)

A partir de la medida de error absoluto se obtiene la SR de un algoritmo. Cuando ejecutamos un algoritmo frente a un problema un número n de ejecuciones, la SR representa el tanto por ciento de ejecuciones que se resuelve. Por lo tanto, a mayor tasa de éxito, mejor es un algoritmo. Esta medida proporciona información muy útil sobre la estabilidad de un algoritmo, es decir, lo fiable que es a la hora de resolver una función. Para calcularla es necesario establecer un nivel de precisión ϵ, es decir, se considera que un algoritmo resuelve una ejecución cuando la solución obtenida alcanza un valor absoluto que es mejor que ϵ. La SR se calcula según la siguiente ecuación:

$$SR = \frac{número\ ejecuciones\ resueltas}{número\ ejecuciones} \tag{4.17}$$

Aunque la SR es una medida intuitiva y simple, no proporciona información sobre la tasa de convergencia de los algoritmos en función del número de llamadas a la función de evaluación que necesita un algoritmo para resolver un problema, es decir, de nuevo, no conocemos lo rápido o lento que es un algoritmo. Además, no tiene en cuenta el error absoluto obtenido, es decir, si por ejemplo consideramos que el nivel de precisión ϵ toma el valor de 10^{-6} y el algoritmo obtiene un error absoluto del orden de 10^{-5}, para determinar la tasa de éxito se considera que dicha ejecución es un fallo cuando, en realidad, esa ejecución puede considerarse *casi* un éxito. De cualquier forma, es una medida útil para estimar de manera simple el número de ejecuciones en las que se ha alcanzado el óptimo en un experimento.

Tasa de rendimiento (SP)

Para resolver el primer problema de la SR, es decir, con el objetivo de considerar la tasa de convergencia para comparar algoritmos, generalmente se utiliza la tasa de rendiemiento (SP). Esta medida tiene en cuenta el número de llamadas a la función de calidad (FEs) que el algoritmo utiliza en cada ejecución hasta alcanzar el nivel de error establecido por el usuario, que generalmente toma el mismo valor que el nivel de precisión para calcular la tasa de éxito, ϵ. Cuando se comparan dos algoritmos en términos de la tasa de rendimiento, el mejor algoritmo es el que obtiene la tasa más baja, ya que es el que menos llamadas a la función de calidad ha necesitado para resolver el problema y, como consecuencia, el que mejor ha explorado el espacio de búsqueda. Esta medida se calcula según la ecuación siguiente:

$$SP = \frac{\overline{FEs} * \textit{ número ejecuciones}}{\textit{número ejecuciones resueltas}} = \frac{\overline{FEs}}{SR} \tag{4.18}$$

donde $\overline{FEs} = FEs_{max}$ cuando el nivel de error absoluto alcanzado por el algoritmo no alcanza el nivel de precisión, ϵ, establecido por el usuario.

Aunque esta medida soluciona algunos de los problemas de las dos medidas anteriores, al utilizar un valor de precisión establecido por el usuario para determinar el valor de FEs, sigue sin tener en cuenta el error absoluto obtenido por los algoritmos.

Medida combinada de error y rendimiento (CPEM)

Con el objetivo de resolver los problemas de las medidas de tasa de éxito y tasa de rendimiento, es decir, que ambas medidas ignoran el error absoluto obtenido por el algoritmo cuando una ejecución se considera que ha fallado (cuando no alcanza el nivel de precisión establecido), se propone una nueva medida que tiene en cuanta ambas características, la tasa de convergencia y el error absoluto obtenido por un algoritmo, ya sea la ejecución un éxito o un fallo. Esta es la CPEM (*Combined error and performance measure*) que se define de la siguiente manera:

$$CPEM = \begin{cases} \epsilon \cdot \dfrac{FEs}{FEs_{max}} & \text{si } FEs \leq FEs_{max} \\ \text{error absoluto} & \text{en otro caso} \end{cases} \tag{4.19}$$

Con esta medida, las ejecuciones que alcanzan el nivel de error establecido son recompensadas. La recompensa es inversamente proporcional el número de llamadas a la función de evaluación que ha utilizado el algoritmo para resolver el problema. Si la ejecución no alcanza el nivel de error, el valor de esta nueva medida es el error absoluto obtenido. Como consecuencia, el resultado de esta medida no es un valor binario éxito o fallo, si no que proporciona valores intermedios que nos permiten utilizar información adicional. En el procedimiento de caracterización propuesto se utiliza esta medida como base para analizar y comparar el comportamiento de los algoritmos seleccionados.

Error genotípico (G_{NE})

Todas las medidas presentadas anteriormente analizan el comportamiento de los algoritmos desde un punto de vista fenotípico. Sin embargo, cuando se trabaja con algoritmos evolutivos, es interesante analizar también el comportamiento de los mismos desde el punto

de vista genotípico, especialmente cuando se trabaja con funciones multimodales. En las funciones multimodales, un algoritmo puede alcanzar un óptimo local con un valor de error absoluto alto pero que se encuentra situado cerca del óptimo global en el espacio de soluciones o un óptimo local con un valor de error absoluto bajo pero que se encuentra lejos del óptimo global. Estas dos situaciones deberían ser detectadas y analizadas. Con este propósito, en este trabajo se utilizará el error genotípico (G_{NE}) para analizar los algoritmos. El G_{NE} se calcula utilizando la siguiente fórmula:

$$G_{NE} = \frac{\sqrt{\sum_{i=0}^{n}(x_i - x_i^*)^2}}{n} \tag{4.20}$$

Esta medida calcula la distancia genotípica entre la solución obtenida por el algoritmo y la mejor solución conocida para el problema.

Resumiendo, las principales características de las medidas de error y rendimiento presentadas son las siguientes:

- El error absoluto es una medida simple y fácil de calcular pero no proporciona información sobre la velocidad de convergencia de los algoritmos.

- La SR es también simple pero, de nuevo, no proporciona información sobre la velocidad de convergencia. Es una medida *binaria* que solamente considera éxitos o fracasos y no proporciona información de resultados intermedios, siendo su mayor utilidad la simplicidad a la hora de estimar el número de ejecuciones que alcanzan el óptimo.

- La SP proporciona información sobre la velocidad de convergencia, pero al establecer un nivel de precisión no tiene en cuenta los *casi* éxitos y no considera el error absoluto obtenido en las ejecuciones.

- La medida *CPEM* tiene en cuenta toda la posible información que puede obtenerse de una ejecución de un algoritmo desde el punto de vista fenotípico: el error absoluto y la velocidad de convergencia. De todas formas, en caso de realizar varias ejecuciones sobre una cierta función, no indicaría el número de ejecuciones acertadas y falladas, ya que es una medida continua.

- El G_{NE} analiza los algoritmos desde el punto de vista del cromosoma y proporciona información sobre lo cerca o lejos que está la solución obtenida de la solución óptima.

El procedimiento de análisis y comparación de algoritmos que será llevado a cabo en el capítulo de resultados de este trabajo será realizado, principalmente, en términos de la medida de error *CPEM*, de esta forma, reduciendo la cantidad de información que se muestra, se simplifica el proceso de análisis. De todas formas, para establecer una estimación inicial de resultados se utilizará también el SR que proporciona información simple sobre cuántas ejecuciones alcanzan el óptimo y cuántas no. Debemos destacar que el SR no debe ser utilizado nunca como medida principal de análisis de resultados. Para este objetivo se usará el CPEM or último, para un análisis más detallado sobre la precisión real de un resultado se utilizará el G_{NE}.

Como conclusión a esta segunda etapa del procedimiento de caracterización se debe destacar la gran importancia que tiene realizar una elección adecuada de las medidas de error y rendimiento que se utilizarán para el análisis de los AEs. Los desarrolladores pueden hacer uso de las medidas establecidas en el marco de este trabajo, o bien proponer otras nuevas, pero siempre incidiendo en que resulten objetivas y en reducir el número de ellas. No se debe olvidar que se deberán analizar gran cantidad de datos, por lo que utilizar una única medida fiable será de gran ayuda.

4.3 Estudio experimental

4.3.1 Ejecución

El primer paso del estudio experimental de un AE consiste en ejecutar dicho algoritmo sobre el conjunto de funciones de prueba. Para realizar este paso, es necesario establecer una serie de condiciones de ejecución de forma que los resultados obtenidos sean generalizables. Estas condiciones de ejecución se dividen en dos tipos: aquellas relacionadas con los parámetros de los AEs y las que hacen referencia explícitamente a las condiciones de ejecución de la implementación concreta del algoritmo.

1. **Parámetros de los AEs:** antes de la ejecución del algoritmo, el diseñador deberá establecer los parámetros del mismo. Dentro de estos parámetros se distingue entre los parámetros propios del AE por un lado y el número de individuos de la población por otro.

    - **Operadores y configuración:** en cuanto a los operadores propios de cada algoritmo y su configuración, este procedimiento de caracterización ha sido desarrollado con el propósito de formalizar la tarea de caracterización y presentación de resultados por parte del diseñador del algoritmo. Por este motivo, previo a la aplicación de este procedimiento, el diseñador habrá determinado qué valores de los parámetros de configuración utilizará en su algoritmo. Se recomienda que estos valores sean aquellos que proporcionen, en promedio, los mejores resultados en problemas de optimización con codificación real. Por lo tanto, no es objeto de este procedimiento la realización de un barrido de los valores de los parámetros de configuración.
    - **Tamaño de la población:** como parte de este procedimiento de caracterización, se recomienda realizar un análisis previo de los requisitos poblacionales del algoritmo presentado. El tamaño de la población es un parámetro que comparten todos los AEs y, además, es altamente dependiente de la dimensionalidad de la función objetivo. Este análisis se recomienda para comparar los resultados obtenidos por diferentes algoritmos en condiciones igualitarias. Para obtener el tamaño de población óptimo en cada función los algoritmos se ejecutarán utilizando un rango de tamaños de poblaciones. Se realizarán tantas ejecuciones del algoritmo como combinaciones existan de funciones de prueba, dimensión del problema y tamaño de población analizado. Tras dicha ejecución se analizarán los resultados proporcionados y se seleccionará aquel que mejores resultados proporcione en términos de *CPEM* utilizando un criterio de parada que será explicado más adelante. Este tamaño de población óptimo es el que será utilizado en un paso

posterior para analizar el comportamiento de los algoritmos. Es importante dejar claro que el algoritmo no volverá a ejecutarse de nuevo, sino que se utilizará el mejor resultado obtenido en este análisis poblacional. Para realizar el barrido se comenzará con un tamaño de población de 10 individuos y se incrementará este valor hasta que el valor de CPEM obtenido empeore con respecto a los anteriores tamaños de población.

2. **Condiciones de ejecución:** para realizar un análisis homogéneo y, sobre todo, equitativo de los AEs es necesario establecer una serie de condiciones de ejecución que los diseñadores deben cumplir para conseguir este objetivo y para que, además, los resultados obtenidos sean comparables con los que obtengan otros algoritmos. Las condiciones de ejecución utilizadas en este trabajo, y que se describen a continuación, son las mismas que se utilizan en la mayor parte de las competiciones de AEs.

   - **Configuración de las ejecuciones:** para cada algoritmo, dimensión y tamaño de población se realizarán 25 ejecuciones independientes (como en [Suganthan et al., 2005]). Como se puede suponer, tal número de ejecuciones implica un alto coste computacional pero hay que tener en cuenta que este estudio se realizará una única vez y que este análisis forma parte del proceso de validación del algoritmo. En cada ejecución deberán ser almacenados los datos de error absoluto, número de evaluaciones necesarias para alcanzar el óptimo y la distancia genotípica entre el óptimo y la solución obtenida. Con esta información se calcularán los valores de *CPEM*, tasa de éxito y error genotípico como la media de los resultados obtenidos en cada una de las 25 ejecuciones y se calculará también la desviación estándar de las 25 pruebas. En este trabajo, para el cálculo de *CPEM* y de la tasa de éxito se ha considerado que una ejecución está resuelta cuando el error absoluto obtenido es menor que 10^{-6}, este valor de precisión es el más utilizado en las competiciones del campo de los AE.
   - **Criterio de parada:** en este trabajo se ha tomado la decisión de utilizar el mismo criterio de parada que se utiliza en la mayoría de las competiciones sobre AEs, es decir, fijar el número máximo de llamadas a la función de evaluación que puede realizar cada algoritmo. En este caso, este número máximo se ha fijado al valor de $10000 \cdot n$, donde n es la dimensión del problema (el mismo criterio que en [Suganthan et al., 2005]). Si el algoritmo analizado alcanza el valor óptimo antes de ejecutar dicho número de evaluaciones, la ejecución finalizará.

Una vez ejecutado el algoritmo sobre el conjunto de funciones de prueba se dispone de una gran cantidad de datos. El primer paso consiste en seleccionar el tamaño de población óptimo para cada función. A continuación, con el tamaño escogido se analizarán los resultados en base a la caracterización del espacio de calidad realizada en la primera etapa del procedimiento. Este análisis de los resultados deberá ser lo más general posible, de modo que se extraigan conclusiones sobre el comportamiento del AE en función de sus características básicas y no de la implementación particular del mismo.

4.3.2 Análisis de resultados

La segunda parte del estudio experimental consiste en analizar los resultados obtenidos tras la ejecución del AE para realizar la caracterización del algoritmo en sí. Este análisis

se debe llevar a cabo para el tamaño de población óptimo en cada función del conjunto de prueba.

En este trabajo se propone una metodología de análisis de resultados que consta de tres pasos consecutivos:

1. Análisis con la configuración básica: como se ha establecido en la sección anterior, el diseñador del AE debe seleccionar los parámetros del mismo (excepto el tamaño de población) que proporcionan una respuesta óptima, en este caso, en problemas de optimización con codificación real.

2. Análisis con modificaciones en los parámetros: aquellas funciones en las que el AE claramente falle con la configuración básica deberán ser tratadas en profundidad. Se deberá estudiar cada caso o grupos de casos por separado estableciendo una posible razón para el fallo, y modificando los parámetros del AE con objeto de solucionar el problema.

3. Comparación con otros algoritmos: una vez que el AE ha sido caracterizado de forma aislada, sus resultados deben ser comparados con los de otros algoritmos caracterizados en los mismos términos de cara a proporcionar un análisis relativo.

La clave para que este análisis resulte general consiste en realizarlo en base a las características intrínsecas que condicionan el funcionamiento de un AE, y que en todas las implementaciones particulares deben estar presentes, aunque se encuentren gobernadas por operadores específicos. Tras un estudio detallado de este tipo de algoritmos, podemos concluir que estas características básicas de todos los AE son cuatro: el balance exploración / explotación, la presión selectiva, las direcciones de búsqueda y el tamaño del paso de mutación.

Exploración vs. Explotación

Los AEs basan su estrategia de búsqueda en dos procesos: la exploración del espacio y la explotación de soluciones prometedoras. Mediante la exploración, los algoritmos evolutivos intentan cubrir el mayor área posible dentro del espacio de búsqueda con el objetivo de encontrar soluciones prometedoras. Consiste, pues, en generar soluciones a partir de grandes "saltos" por el espacio de búsqueda. Es un mecanismo muy útil para evitar que la población converja hacia óptimos locales manteniendo un alto grado de diversidad. Por el contrario, la explotación tiende a generar soluciones próximas a otras ya existentes haciendo pequeñas modificaciones sobre ellas.

El principal problema de los algoritmos evolutivos es encontrar un conjunto de parámetros que consigan un balance adecuado de exploración y explotación que evite el estancamiento de la población y la convergencia prematura. El estancamiento de la población se produce cuando no existe una mejora de la calidad media o del mejor individuo de la población durante un número de generaciones dado. La convergencia prematura ocurre cuando la población tiende hacia un óptimo local debido, entre otras causas, a la falta de diversidad en la población. El proceso de búsqueda de un algoritmo con demasiada exploración es parecido al de una búsqueda aleatoria. La población tendrá tendencia a caer en un proceso cíclico en el cual nunca converja hacia ningún óptimo y se generen siempre nuevas soluciones que no son explotadas. Si, por el contrario, el algoritmo evolutivo presenta demasiada explotación, la población tenderá a converger hacia óptimos locales.

En numerosos trabajos se han realizado análisis sobre la influencia de los operadores concretos en el balance exploración / explotación. Los operadores de mutación son los encargados de introducir nueva información en la búsqueda, los de cruce se encargan de combinar la información ya conocida y, por último, los operadores de selección y reemplazo se encargar de dirigir el proceso de búsqueda a zonas prometedoras y de conservar la diversidad en la población para evitar la convergencia prematura de la población. La caracterización de los operadores se realiza en base a su contribución en el balance exploración / explotación. En [Eiben and Schippers, 1998] los autores realizaron una revisión sobre este tema, concluyendo que no hay un consenso generalizado. La mayoría de los autores están de acuerdo en que los operadores de cruce exploran y el operador de mutación tiene tendencia a la explotación. Pero si se analiza más en profundidad se obtienen las siguientes conclusiones:

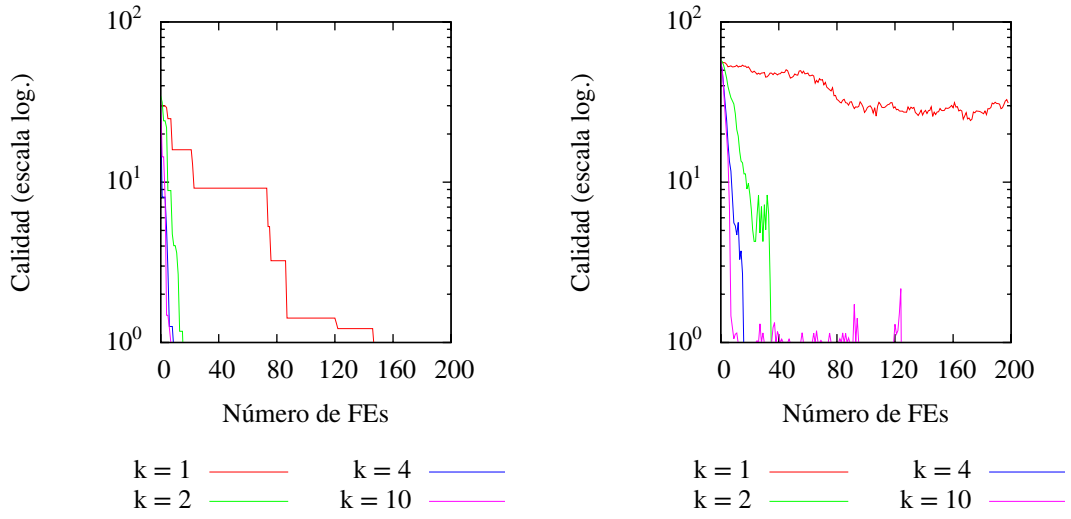
- La mutación puede verse como un operador de exploración ya que introduce nuevo material, generalmente aleatorio, a las soluciones de la población. Pero también puede verse como un operador de explotación debido a que es un operador muy conservador al realizar pequeños cambios y la mayor parte de la información del padre se conserva.
- El cruce puede considerarse un operador de exploración porque combina información de dos padres, es decir, dos configuraciones distintas y se generarán soluciones como combinación de diferentes hiperplanos del espacio de búsqueda, es decir, de zonas que no han sido "visitadas" anteriormente. Aunque, desde otro punto de vista, puede ser considerado un operador de explotación porque utiliza información "antigua" proporcionada por los padres para generar la descendencia.
- Si el algoritmo utiliza operadores multi-parentales las propiedades de exploración y de explotación se intensifican porque hay más información (en forma de "padres") para generar las nuevas soluciones.
- Los operadores de selección y reemplazo son una fuente de explotación ya que no introducen nueva información en la búsqueda, únicamente utilizan información conocida para dirigir el proceso de optimización.

Por norma general, los AEs tienden a comenzar el proceso evolutivo con una fase altamente exploradora. De esta forma, durante las primeras fases se realiza una especie de reconocimiento global del espacio de búsqueda. A medida que se avanza en el proceso evolutivo se tiende a una estrategia explotadora, en la cual se generan pequeñas modificaciones sobre las mejores soluciones encontradas durante la fase exploradora. Pero lo bueno o malo que es este balance tiene una gran dependencia del problema a resolver como veremos en capítulos posteriores.

Por tanto, a la hora de analizar el comportamiento de un AE, se debe estudiar el balance exploración / explotación de sus operadores concretos y, además, cómo afecta la configuración de sus parámetros a este balance.

Presión selectiva

El concepto de presión selectiva está relacionado con los operadores de selección y su influencia en el balance exploración / explotación. Hace referencia a las probabilidades de seleccionar a los mejores individuos de la población. Es decir, un operador con alta presión selectiva hace que los mejores individuos sean elegidos con mucha mayor probabilidad reduciendo rápidamente la diversidad en la población. Por el contrario, un operador con baja



(a) Evolución de la calidad del mejor individuo. (b) Evolución de la calidad media de la población

Figura 4.22: Representación de cómo afecta la presión selectiva a la evolución en un AG con diferentes configuraciones.

presión selectiva hace que todos los individuos de la población tengan las mismas posibilidades de ser elegidos. Por tanto, individuos con baja calidad también podrán sobrevivir generación tras generación. Para medir el grado de presión selectiva de un operador se utiliza típicamente el *takeover time* [Bäck, 1994]. Mide el tiempo que tardaría una población en converger hacia una única solución si solamente se utilizase el operador de selección.

En la figura 4.22 se muestra como afecta la presión selectiva al proceso de evolución. En dichas imágenes se presenta la evolución de la calidad de un algoritmo de tipo AG en el cual se utiliza un operador de selección por torneo. Utilizando este operador es posible balancear la presión selectiva modificando el tamaño de la ventana de selección. A medida que crece el tamaño de la ventana aumenta la presión selectiva, en este caso se han utilizado tamaños de ventana 1, 2 , 4 y 10. En las figuras se muestra la evolución de la calidad del mejor individuo (figura 4.22a y la evolución de la calidad media de la población 4.22b. Cuando el tamaño de la ventana de selección es bajo (k = 1) la velocidad de convergencia en el algoritmo es menor. El algoritmo necesita más generaciones para alcanzar la solución óptima y la calidad media de la población nunca converge a la calidad del mejor, es decir, existe mucha diversidad en la población. A medida que se aumenta el tamaño de la ventana crece la presión selectiva y, por lo tanto, la velocidad de convergencia. En estos casos (k = 2 y k = 4), la calidad media se aproxima más a la calidad del mejor individuo, lo cual provoca una falta de diversidad. Si el tamaño de la ventana se incrementa hasta 10 individuos nos encontramos con un caso de muy alta presión selectiva donde las probabilidades de ser seleccionado cuando el individuo tiene baja calidad son muy bajas. En casos como este, el algoritmo tiene muchas probabilidades de converger rápidamente hacia un óptimo local, como ocurre aquí.

En la tabla 4.6 se muestran los resultados numéricos de los experimentos de la figura 4.22. Estos resultados ratifican lo que ha sido comentado anteriormente. Para realizar estas pruebas se ha utilizado como criterio de parada el número máximo de llamadas a la función de evaluación, que en este caso se estableció a 100000. Cuando la presión selectiva es baja (k = 1), la velocidad de convergencia es muy baja y la población no converge al óptimo como se puede ver en la diferencia que existe entre la calidad media de la población y la

Tamaño ventana (k)	Mejor	Media	FEs
1	0.487	31.185	-
2	0.000	0.000	5200
4	0.000	0.000	2300
10	0.099	0.099	-

Tabla 4.6: Cómo afecta la presión selectiva a la evolución en un AG con diferentes configuraciones.

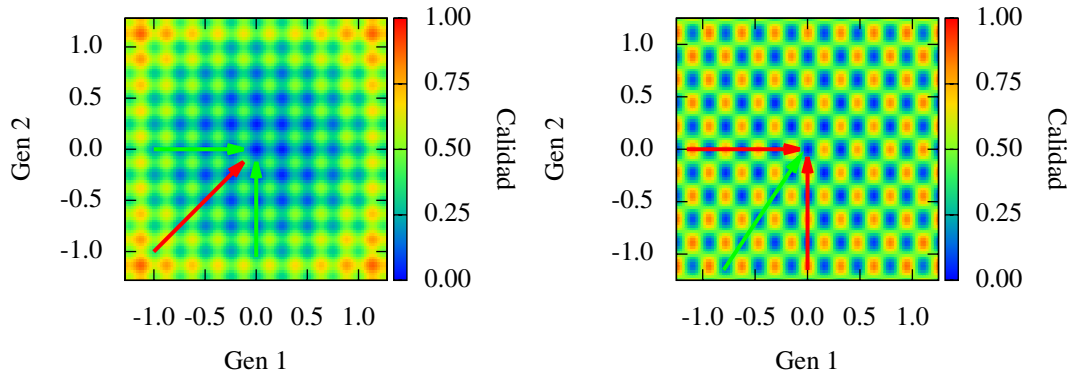
calidad del mejor individuo. Por lo tanto, el grado de diversidad es elevado. El algoritmo no es capaz de encontrar la solución óptima al problema (establecida en 10^{-6}) debido a esa gran diversidad, el nivel de exploración es elevado y el algoritmo se comporta como una búsqueda aleatoria sin dirigir su búsqueda hacia las soluciones consideradas buenas. Al aumentar el tamaño de la ventana ($k = 2$ y $k = 4$) aumenta también el nivel de explotación y el algoritmo tiende a dirigir su búsqueda hacia las zonas de mejor calidad. En estos dos casos el algoritmo resuelve el problema siendo más rápido (2300 llamadas a la función de evaluación frente a 5200) cuando la ventana es de mayor tamaño debido a que la presión selectiva es mayor. En el caso extremo, $k = 10$, la presión selectiva es tan elevada que el algoritmo tiene tendencia a converger hacia óptimos locales, como ocurre en este caso en el cual toda la población ha convergido hacia el mismo óptimo local como se puede observar comprobando que el valor de calidad del mejor individuo es el mismo que el valor de calidad medio.

Por tanto, otra propiedad básica de los AEs es la presión selectiva. El nivel de la misma se ajusta a partir de los operadores de selección y reemplazo. Un buen ajuste de dicha presión es importante tanto en las funciones multimodales como en las unimodales. En las primeras porque afecta a la tendencia a caer en óptimos locales. En las segundas debido a la velocidad de convergencia de los algoritmos.

Direcciones de búsqueda

Los AEs son algoritmos de optimización que en una misma generación modifican varios parámetros de los cromosomas de los individuos. De esta forma, el algoritmo es capaz de realizar la búsqueda en más de una dimensión en un solo paso, reduciendo el número de generaciones necesarias. Cuando un algoritmo modifica un solo parámetro en una generación se dice que realiza la búsqueda en direcciones ortogonales a los ejes de coordenadas. En caso contrario, la búsqueda se realiza en direcciones diagonales. Existen problemas en los que la búsqueda óptima se realiza en direcciones ortogonales y funciones en las que el proceso de optimización es más rápido en diagonales. Por ejemplo, en la figura 4.23 se muestran dos espacios de calidad en los cuales las direcciones de búsqueda óptimas son diferentes. En el caso de la figura 4.23a, las direcciones de búsqueda óptimas (flechas de color verde) se corresponden con las direcciones perpendiculares a los ejes de coordenadas. Si el algoritmo de búsqueda modifica dos parámetros al mismo tiempo, buscando en direcciones diagonales (flecha de color rojo), su rendimiento será menor ya que el camino hacia el óptimo es más accidentado. La figura 4.23b muestra el caso contrario, cuando la dirección óptima de búsqueda es diagonalmente (flecha de color verde) y las búsquedas en direcciones ortogonales dificultan el proceso de optimización (flechas en color rojo).

En primer lugar, es interesante conocer qué tipo de búsqueda es la que más favorece a las funciones con las que tratamos. Generalmente, una función separable se beneficia de una



(a) Búsqueda en direcciones ortogonales.

(b) Búsqueda en direcciones diagonales.

Figura 4.23: Representación de dos espacios de calidad con direcciones de búsqueda óptimas diferentes.

búsqueda en direcciones ortogonales, debido a que este tipo de búsqueda realiza procesos de optimización independientes para cada variable, modificando pocos parámetros en cada generación. Si crece el nivel de interacción entre los parámetros de las funciones es más adecuada una búsqueda en direcciones diagonales donde se modifique más de un parámetro en cada generación. Por lo tanto, es interesante conocer si el algoritmo que vamos a utilizar permite balancear y ajustar las direcciones de búsqueda.

Tamaño del paso de mutación

Determinar cuánto se modifica una variable en cada generación del algoritmo es también un aspecto importante dentro del funcionamiento de los AEs. Esta modificación se denomina paso de mutación. El ajuste del mismo afecta tanto a funciones unimodales como multimodales. En el caso de las funciones unimodales afecta a la velocidad de convergencia. Durante los primeros pasos de la optimización un paso de mutación alto ayuda a recorrer más espacio de calidad en menos pasos. Una vez que se alcanza la zona del óptimo es más adecuado un paso de mutación bajo para hacer un ajuste fino hasta alcanzar el punto óptimo. Cuando existe más de un óptimo local, el paso de mutación sirve para salir de óptimos locales hacia los que haya convergido el algoritmo ya que, un paso de mutación alto provocaría un "salto" en el espacio de calidad que permitiría a la solución salir de ese óptimo local.

Existen algoritmos que permiten ajustar el paso de mutación durante el proceso evolutivo. Este ajuste está relacionado con el balance exploración / explotación. Generalmente, durante las primeras etapas el paso de mutación es alto para permitir que se explore el espacio de calidad. A medida que pasan las generaciones, el paso de mutación decrece para permitir el ajuste fino, es decir, una explotación de las soluciones que se han considerado prometedoras.

En la figura 4.24 se muestra el comportamiento de dos algoritmos en dos tipos de espacios de calidad diferentes y cómo son los pasos de mutación en cada uno de los escenarios. Las figuras 4.24a y 4.24b representan el comportamiento de dos AEs en un espacio de calidad unimodal. En las primeras generaciones el paso de mutación es largo para explorar mejor el espacio de calidad, a medida que pasan las generaciones el tamaño del paso decrece para

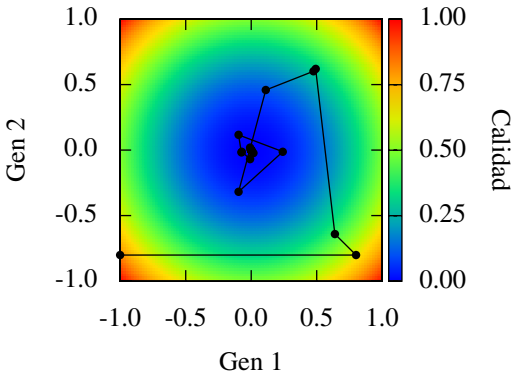
realizar ajustes finos en las soluciones que permitan alcanzar la solución óptima. En las funciones unimodales el ajuste del paso de mutación afecta, principalmente, a la velocidad de convergencia. El algoritmo de la izquierda (figura 4.24a) realiza un mejor ajuste del paso de mutación ya que ejecuta menos pasos para alcanzar el óptimo. En las figura 4.24c y 4.24d el espacio de calidad es multimodal. En este caso el ajuste del paso de mutación ayuda a salir de óptimos locales en caso de que el AE converja hacia uno de ellos. Como se ve en estas figuras los algoritmos "saltan" de óptimo local a óptimo local hasta alcanzar el óptimo de la función.

Las cuatro características explicadas en esta sección son intrínsecas a todos los AEs y, por tanto, el análisis del comportamiento de los mismos deberá realizarse relacionando las propiedades de los espacios de calidad seleccionadas como más relevantes (aquí separabilidad y modalidad) con estas cuatro características. En este sentido, a continuación se resumen algunas relaciones conocidas entre estas propiedades y características, y que serán utilizadas en el segundo paso de la metodología de análisis propuesta a la hora de decidir qué parámetro modificar para solucionar un problema del AE:

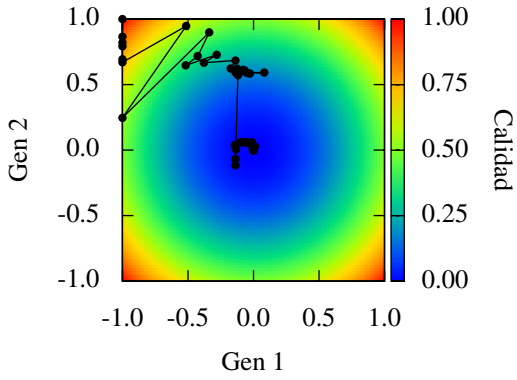
- Los algoritmos con un mayor nivel de exploración se recomiendan para resolver espacios de calidad multimodales con muchos óptimos locales y con amplias distancias entre ellos.
- Por el contrario, un algoritmo con un alto nivel de explotación es más aconsejable cuando la función es unimodal, ya que estas funciones se benefician de una mayor velocidad de convergencia.
- Para resolver una función separable es aconsejable un algoritmo que busque en direcciones ortogonales. Por el contrario si la función es no separable se recomienda un algoritmo que busque en diagonales.
- El concepto de presión selectiva y el balance exploración / explotación están relacionados: a mayor presión selectiva mayor nivel de explotación y menor nivel de exploración.
- El ajuste del paso de mutación también depende del tipo de función con la cual se quiera tratar y es aconsejable que se adapte automáticamente a la superficie del espacio de calidad.

4.4 Resumen del procedimiento de caracterización

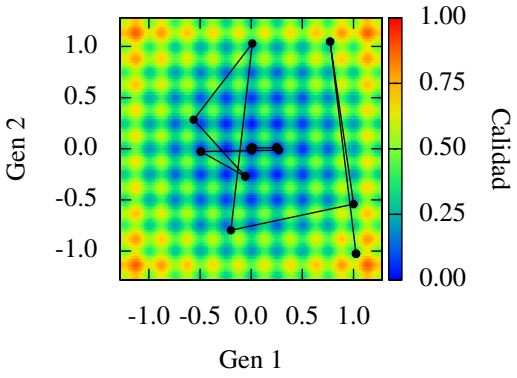
En este capítulo se ha propuesto un procedimiento de caracterización de algoritmos evolutivos que consta de tres etapas interrelacionadas. Estas tres etapas corresponden a los tres tipos de problemas que más han sido tratados en la bibliografía del campo a la hora de aportar soluciones a la falta de estandarización y formalización en el análisis de los algoritmos: la correcta elección del conjunto de funciones prueba, la determinación de las medidas de análisis representativas y el establecimiento de una metodología de ejecución de las pruebas y de análisis de los resultados. Los desarrolladores de algoritmos deben proponer soluciones concretas a cada uno de estos problemas si quieren llevar a cabo una caracterización formal. A lo largo de este capítulo se han desarrollado posibles soluciones que dan aplicabilidad práctica al procedimiento (ver figura 4.25), y que pasamos a resumir a continuación:



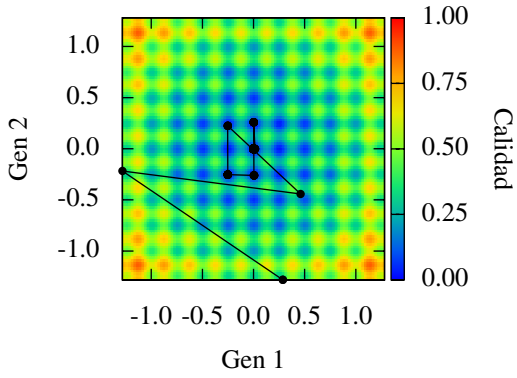
(a) Función unimodal, algoritmo DE



(b) Función unimodal, algoritmo GA



(c) Función multimodal, algoritmo DE



(d) Función multimodal, algoritmo GA

Figura 4.24: Representación de diferentes modelos de modificación del paso de mutación. En estos ejemplos se utiliza un AG y un algoritmo DE en dos tipos de funciones: una función unimodal y otra función multimodal.

1. **Caracterización de espacios de calidad**. En este paso los desarrolladores tienen dos alternativas:

   1.1. Obtener el conjunto de funciones prueba que se describe en el apéndice B y que ha sido previamente caracterizado en términos de separabilidad y modalidad como puede verse en la sección 4.1.3 utilizando los tres algoritmos presentados en dicha sección. Este conjunto de funciones de prueba está accesible en la web de Java Evolutionary Algorithm Framework (JEAF) (http://www.gii.udc.es/jeaf).

   1.2. Utilizar otro conjunto de funciones de prueba diferente. En este caso, el desarrollador deberá estimar las propiedades de estas funciones utilizando los tres algoritmos presentados en este capítulo.

2. **Medidas de error y rendimiento**. Se propone utilizar, principalmente, la medida *CPEM* que combina información sobre error y rendimiento en un solo dato. Además, se deberán computar también las medidas de *SR* y G_{NE} para apoyar las conclusiones que se obtengan. La primera de ellas sirve para estudiar los algoritmos en cuanto a estabilidad, en términos de cuántas ejecuciones resuelven. La segunda permite analizar la precisión del algoritmo en una cierta función. Por tanto, los desarrolladores deberán añadir a su algoritmo el cálculo estas tres medidas de error para su posterior análisis.

3. **Estudio experimental**. Tras seleccionar y caracterizar el conjunto de funciones de prueba, y tras establecer las medidas que se utilizarán para analizar los resultados, el desarrollador deberá realizar el proceso experimental consistente en ejecutar el algoritmo y estudiar los resultados obtenidos.

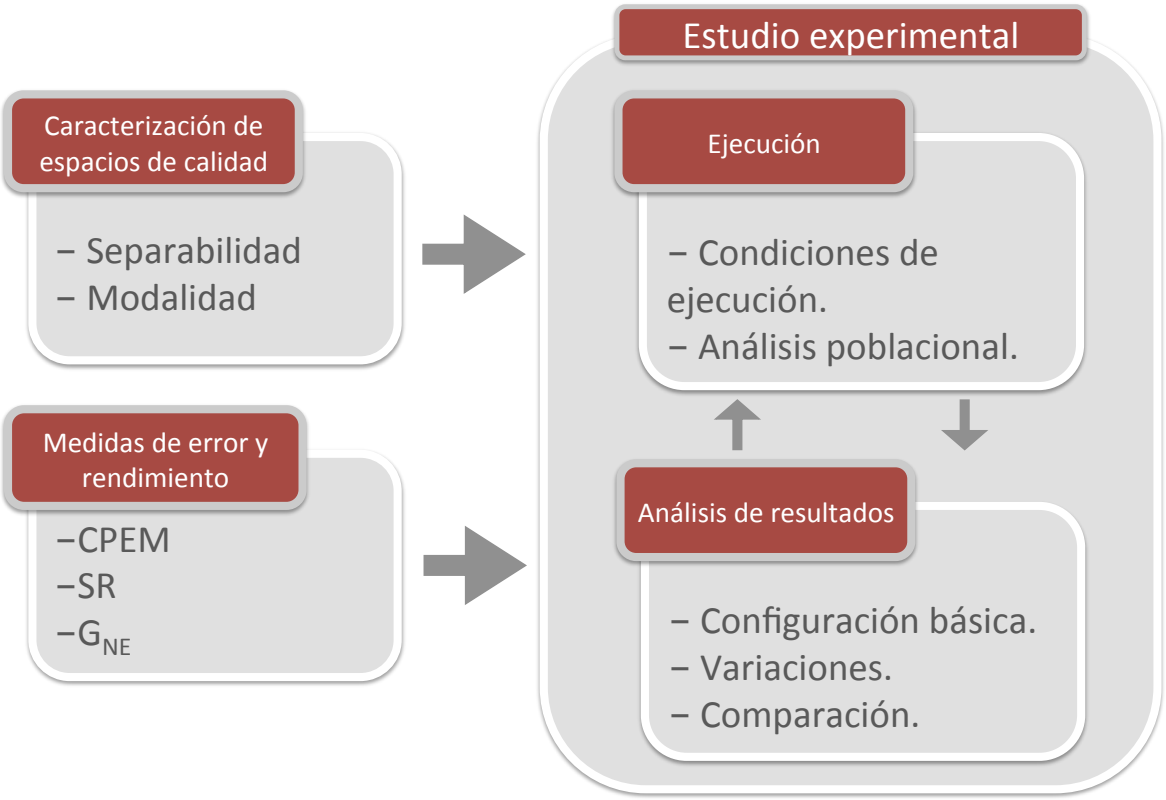


Figura 4.25: Elementos del procedimiento de caracterización propuesto en este trabajo.

3.1. **Ejecución**. El desarrollador deberá ejecutar el AE sobre el conjunto de funciones prueba seleccionado en el paso 1 siguiendo las condiciones de ejecución que fueron explicadas en la sección 4.3.1 y que son:

3.1.1. Selección por parte del diseñador de los parámetros propios de su AE que proporcionan un funcionamiento óptimo, a excepción del tamaño de población que lo fija el procedimiento.

3.1.2. El tamaño de población se debe seleccionar tras la realización de un estudio exhaustivo consistente en la ejecución del algoritmo sobre todas las funciones de prueba realizando un barrido en poblaciones que comienza con 10 individuos y se incrementa en 10 mientras el valor de CPEM mejore.

3.1.3. Para cada algoritmo y tamaño de población se realizarán 25 ejecuciones independientes por función (y dimensión en caso de ser una función escalable). En cada caso se calcularán los valores de CPEM, tasa de éxito y error genotípico como la media de los resultados obtenidos en las 25 ejecuciones. Se calculará también la desviación estándar de las 25 pruebas.

3.1.4. Se fijará como criterio de parada el número máximo de llamadas a la función de evaluación en $10000 \cdot n$, donde n es la dimensión de la función. Si el óptimo se alcanza antes de llegar al límite, la ejecución finalizará.

3.2. **Análisis de resultados**. El desarrollador deberá analizar los resultados de todas las pruebas realizadas. Para ello, deberá tener en cuenta que se considera que se ha alcanzado el óptimo si el error absoluto es inferior a 10^{-6}. Una primera etapa del análisis consiste en seleccionar el tamaño de población óptimo para cada función. Tras esta etapa, deberá analizar el comportamiento de su AE en tres fases:

3.2.1. Configuración básica de parámetros: tratará de extraer conclusiones generales sobre los tipos de funciones que resuelve y los que no con la configuración óptima de parámetros seleccionada. Para ello, se recomienda comenzar por analizar la respuesta en funciones L-separables o NL-separables frente a las no-separables. A continuación se puede analizar la respuesta en unimodales y multimodales para, finalmente, realizar un análisis cruzado en cuanto a las características básicas. Una vez establecidas unas pautas de comportamiento, se deberán relacionar con las propiedades particulares del AE en cuanto a su balance exploración/explotación, presión selectiva, direcciones de búsqueda y tamaño del paso de mutación.

3.2.2. Modificaciones a la configuración básica: una vez analizado el comportamiento del algoritmo con la configuración básica, se deberán analizar con detalle aquellos tipos de funciones en los que falla con mayor claridad y sobre estos casos se realizarán modificaciones en los parámetros del AE para tratar de determinar si es un problema solucionable o es un fallo intrínseco al algoritmo.

3.2.3. Comparación con otros algoritmos: los resultados obtenidos por el algoritmo deberán ser comparados con los de otros caracterizados previamente utilizando este mismo procedimiento de caracterización de cara a relativizar las conclusiones iniciales.

El siguiente capítulo está dedicado a la aplicación de este procedimiento de caracterización. Para ello se utilizarán cuatro AEs sobre los que serán aplicados los pasos anteriores realizando una caracterización formal de los mismos en términos de separabilidad y modalidad.

Capítulo 5

Aplicación del procedimiento de caracterización

Este capítulo se centra en la aplicación del procedimiento de caracterización descrito en el capítulo anterior. Se trata, fundamentalmente, de demostrar que la utilización de un procedimiento formal como el propuesto proporciona una información sobre los AEs que resulta de gran relevancia para su utilización por parte de los usuarios. Para ello, se han seleccionado cuatro algoritmos. En primer lugar se estudiará un algoritmo genético con codificación real (RCGA), fundamentalmente por ser el algoritmo más utilizado en el campo, el más conocido por parte de los usuarios y una referencia de gran importancia a la hora de comparar AEs. En los apartados siguientes se analizarán los algoritmos Covariance Matrix Adaptation (CMA-ES) y Differential Evolution (DE) por ser los más referenciados en la actualidad, ya que tanto en competiciones como en estudios comparativos han demostrado ser los más potentes y rápidos. Finalmente, el cuarto apartado estará centrado en la caracterización del algoritmo algoritmo macroevolutivo (MA), un algoritmo menos utilizado que los tres anteriores y que precisamente por eso ha sido seleccionado. Este algoritmo permitirá comprobar las consecuencias de una caracterización formal como la que se propone sobre un algoritmo nuevo y poco estudiado.

Para poder llevar a cabo la caracterización de estos cuatro algoritmos se aplicarán las soluciones concretas propuestas en el capítulo anterior en cada una de las tres etapas del procedimiento. En consecuencia, las dos primeras etapas son comunes a los cuatro algoritmos:

1. Selección y caracterización del conjunto de funciones de prueba: se utilizará, como conjunto de funciones prueba, el conjunto de funciones caracterizado en la sección 4.1.2 cuyas propiedades se resumen en las tablas 4.3, 4.4 y 4.5. Por tanto, los cuatro algoritmos serán caracterizados en terminos de separabilidad y modalidad.

2. Selección de las medidas de error y rendimiento: en este apartado los resultados serán analizados utilizando como medidas de error y rendimiento el CPEM, el SR y el G_{NE}. Estas medidas servirán para caracterizar el algoritmo en términos de rendimiento, estabilidad y precisión.

La tercera etapa de estudio experimental consta de la ejecución y el análisis de resultados, es particular para cada algoritmo, y se describe con detalle en cada apartado del capítulo. Estos apartados comienzan con una descripción del algoritmo a caracterizar incidiendo en la influencia de sus parámetros sobre la estrategia de búsqueda. El análisis de los algoritmos se realiza, en primer lugar, sobre su configuración básica y, a continuación, modificando dicha

configuración básica para las funciones en las que se comporta erróneamente. Finalmente, se analizarán los cuatro algoritmos propuestos comparativamente, de tal forma que los resultados obtenidos sean relativos y no absolutos, proporcionando así una caracterización práctica para los usuarios.

5.1 Algoritmo genético para problemas de codificación real (RCGA)

5.1.1 Descripción del algoritmo

Los algoritmos genéticos (AG) son la base de los algoritmos evolutivos y fueron presentados inicialmente por John Holland y David Goldberg [Holland, 1975, Goldberg, 1989a]. La idea básica de estos algoritmos es mantener una población de cromosomas, los cuales representan soluciones candidatas a un problema, que se evolucionan durante un proceso de competición y variación controlada. Los AGs presentan una tasa de éxito muy alta en problemas de búsqueda y optimización.

Desde su aparición, los cromosomas de longitud fija y de codificación binaria han dominado los desarrollos en el campo de los AGs, ya que diversos estudios teóricos han demostrado su eficiencia y, además, la implementación es muy simple. Sin embargo, para muchos problemas es necesario, y también natural, representar los genes de manera directa como números reales. Estos problemas utilizan parámetros en dominios continuos y en ellos cada gen representa una variable del problema. Utilizando parámetros de codificación real en vez de binaria se dispone de dominios más amplios para las variables. Otra ventaja, cuando se utiliza representación real, es su capacidad para explotar los cambios graduales de las funciones, es decir que para pequeños cambios en las variables produzcan pequeños cambios en la función de calidad.

En el ámbito de la codificación real, los operadores más importantes son los de mutación y los de cruce, ya que el operador de selección no depende de la codificación del cromosoma. Existen multitud de operadores de cruce y mutación para parámetros reales en AGs. En [Herrera et al., 1998, Herrera et al., 2003] los autores realizan una revisión formal de los mismos. A la vista de las conclusiones aportadas por estos autores, en este trabajo se ha implementado un AG con un cruce de tipo BLX-α [Eshelman and Schaffer, 1992] y la mutación no uniforme de Michalewicz [Michalewicz, 1994]. A continuación se explican ambos operadores.

Cruce BLX-α Sean $C_1=(c_1^1,\ldots,c_n^1)$ y $C_2=(c_1^2,\ldots,c_n^2)$ dos cromosomas que han sido seleccionados para aplicar sobre ellos el operador de cruce. Aplicando el operador BLX-α se genera un nuevo individuo $H=(h_1,\ldots,h_n)$, donde h_i es un número seleccionado de forma aleatoria utilizando una distribución uniforme dentro del intervalo $[c_{min}\text{-}I\cdot\alpha,\ c_{max}+I\cdot\alpha]$, siendo $c_{min} = \min(c_i^1,c_i^2)$, $c_{max} = \max(c_i^1,c_i^2)$ e $I=c_{max} - c_{min}$.

En la figura 5.1 se muestra de forma esquemática el comportamiento de este operador. El valor de α se utiliza para determinar los límites de este intervalo. Puede tomar tanto valores negativos como positivos. Cuando $\alpha \leq 0$ el nivel de explotación es máximo, los individuos se generan entre los dos padres haciendo que la población pierda diversidad y tendiendo hacia la convergencia prematura. En caso contrario, cuando $\alpha > 0$, aumenta el nivel de exploración y aumenta la diversidad en la población. Para conseguir un balance ideal entre exploración

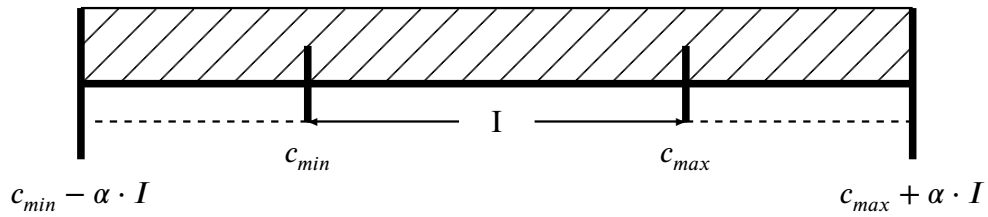


Figura 5.1: Representación del funcionamiento del cruce BLX-α, todos los puntos en el intervalo [c_{min}-I·α, c_{max}+I·α] tienen la misma probabilidad de ser escogidos.

y explotación, en [Herrera et al., 1998] los autores recomiendan un valor para α de 0.5. De esta forma, la probabilidad de generar un individuo entre los dos padres es la misma que la de generarlo fuera de ellos.

Mutación no uniforme de Michalewicz Si C=(c_1,…,c_n) es un cromosoma y c_i ∈[a_i,b_i] un gen que va a ser mutado, entonces:

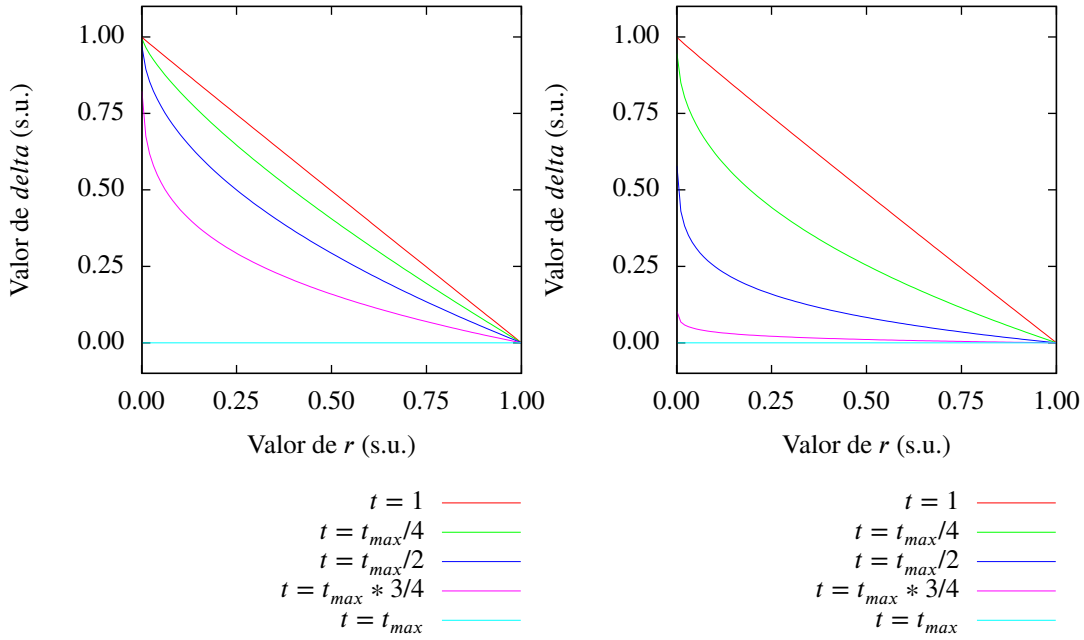
$$c_i' = \begin{cases} c_i + \delta(t, b_i - c_i) & \text{si} \quad \tau = 0 \\ c_i - \delta(t, c_i - a_i) & \text{si} \quad \tau = 1 \end{cases} \tag{5.1}$$

siendo t la generación actual del algoritmo, τ un número aleatorio cuyo valor puede ser 0 o 1, y δ:

$$\delta(t, y) = y \cdot \left(1 - r^{\left(1 - \frac{t}{t_{max}}\right)^b}\right) \tag{5.2}$$

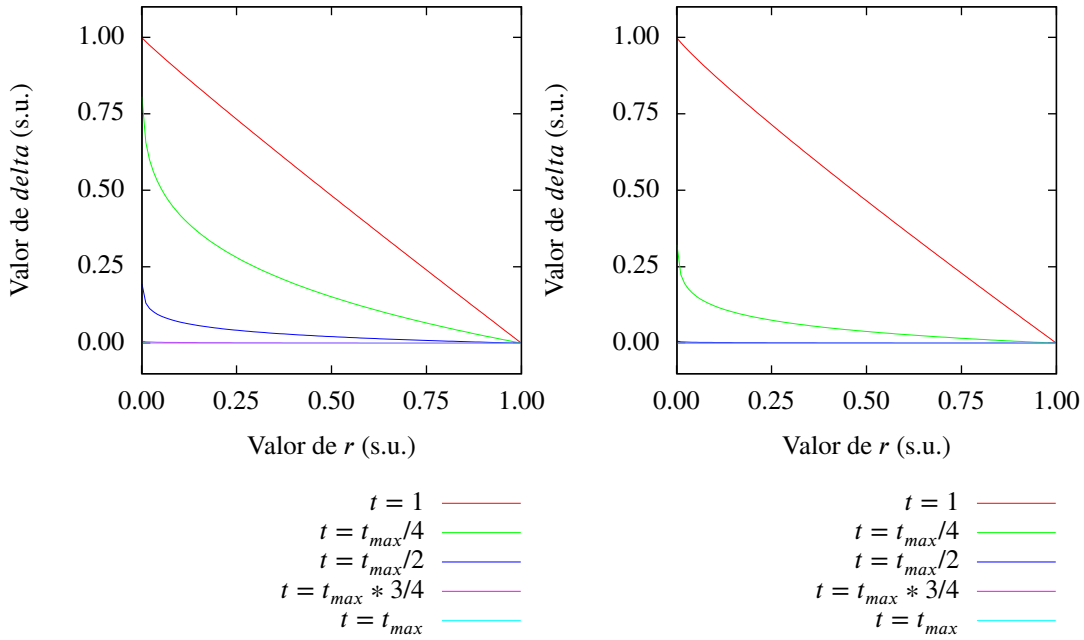
donde t_{max} es el número máximo de generaciones del algoritmos genéticos de codificación real (RCGA), r es un número aleatorio que pertenece al intervalo [0,1] y b es un parámetro establecido por el usuario que determina el grado de dependencia del operador con el número de generaciones transcurridas. La función δ devuelve un valor del intervalo [0,y]. La probabilidad de que el valor devuelto sea cero crece a medida que avanzan las generaciones del algoritmo. El operador hace una búsqueda uniforme en el espacio en las primeras generaciones. A medida que estas avanzan se favorece la búsqueda local. Este comportamiento se refleja en las imágenes de la figura 5.2. En ellas se muestran los posibles valores devueltos por la función δ dependiendo del valor de r, del parámetro b y del instante de la evolución en el que nos encontremos. Con respecto la generación t en la cual se encuentre el algoritmo, como se puede observar en las cuatro gráficas, la probabilidad de obtener un valor cercano a 0 crece a medida que pasan las generaciones. En cuanto al valor del parámetro b, cuando este es bajo (véase las figuras 5.2a y 5.2b) la probabilidad de generar valores cercanos a 0 decrece más lentamente que cuando el valor de b es alto (véase las figuras 5.2c y 5.2d). Esto se traduce en una mayor tendencia hacia la exploración de zonas no conocidas cuando los valores de b son bajos y, por el contrario, una mayor tendencia hacia la explotación de soluciones conocidas cuando los valores de b son altos. El valor de b también afecta al tamaño del paso de mutación, valores bajos de b permiten pasos de mutación más largos que cuando se trabaja con valores altos de este parámetro.

Por tanto, en este trabajo se caracterizará el comportamiento de un AG para problemas de codificación real (RCGA). El pseudocódigo del mismo se presenta en el algoritmo 3. Además de los operadores de cruce y mutación explicados anteriormente, este algoritmo utiliza un operador de selección de tipo torneo con tamaño de ventana 2 y un operador de reemplazo con elitismo. Los cromosomas se codifican en el rango [-1.0:1.0]. La implementación de este algoritmo está diponible en la librería JEAF (www.gii.udc.es/jeaf).



(a) Parámetro b = 1

(b) Parámetro b = 3



(c) Parámetro b = 5

(d) Parámetro b = 10

Figura 5.2: Distintas situaciones para distintos valores del parámetro b en determinados instantes de la evolución. El eje x representa los valores que puede tomar la variable r. El eje y representa el valor de la función δ.

Algoritmo 3 Esquema básico de un Algoritmo Genético

Entrada:

NP, número de individuos de la población.
D, número de genes de cada cromosoma.
t_{max}, número máximo de generaciones.
p_c, probabilidad de cruce.
p_m, probabilidad de mutación.

1: $t \leftarrow 0$
2: Generar la población inicial de forma aleatoria: P(t) = $\{\mathbf{x}_t^i, \forall i, i = 1, \ldots, NP\}$
3: Evaluar: P(t) = $\{\mathbf{x}_t^i, \forall i, i = 1, \ldots, NP\}$
4: **while** $t < t_{max}$ **do**
5: $t \leftarrow t + 1$
6: Seleccionar $P(t)$ a partir de $P(t-1)$
7: **for** $i = 0$ hasta $NP/2$ **do**
8: **if** $p_c \leq rand[0,1)$ **then**
9: $\mathbf{x}_t^{2\cdot i}, \mathbf{x}_t^{2\cdot i+1} = \text{cruce}(\mathbf{x}_{t-1}^{2\cdot i}, \mathbf{x}_{t-1}^{2\cdot i+1})$
10: **else**
11: $\mathbf{x}_t^{2\cdot i}, \mathbf{x}_t^{2\cdot i+1} = \mathbf{x}_{t-1}^{2\cdot i}, \mathbf{x}_{t-1}^{2\cdot i+1}$
12: **end if**
13: **end for**
14: **for** $i = 0$ hasta NP **do**
15: **for** $j = 0$ hasta D **do**
16: **if** $p_m \leq rand[0,1)$ **then**
17: $x_{j,t+1}^i = \text{mutacion}(x_{j,t}^i)$
18: **else**
19: $x_{j,t+1}^i = x_{j,t}^i$
20: **end if**
21: **end for**
22: **end for**
23: Evaluar P(t)
24: **end while**

5.1.2 Caracterización del RCGA

Como se comentó en la introducción del capítulo, partimos de que las dos primeras etapas del procedimiento de caracterización ya han sido llevadas a cabo y se ha seleccionado el conjunto de funciones prueba caracterizado en la sección 4.1.2 cuyas propiedades se resumen en las tablas 4.3, 4.4 y 4.5. En cuanto a las medidas de error y rendimiento, se utilizarán el CPEM, el SR y el G_{NE}. Describimos a continuación la tercera etapa del procedimiento propuesto, el estudio experimental, que consta de ejecución y análisis de resultados.

Ejecución de las pruebas

Lo primero que se debe hacer es ejecutar el algoritmo sobre el conjunto de prueba siguiendo las indicaciones explicadas en 4.3.1. En la tabla 5.1 se muestran los parámetros escogidos para la configuración del RCGA siguiendo las indicaciones de [Herrera et al., 1998]. De esta forma, el algoritmo se configura para proporcionar un rendimiento contrastado en problemas de optimización con parámetros reales.

Operador	Tipo	Parámetro	Valor
Selección	Torneo	Tamaño de ventana	2
Cruce	BLX-α	Probabilidad	60%
		Valor de α	0.5
Mutación	No uniforme de Michalewicz	Probabilidad	0.5%
		Valor de b	5

Tabla 5.1: Parámetros de configuración utilizados para la caracterización del RCGA.

El único parámetro que no prefija el desarrollador es el tamaño de población, que se debe realizar como parte de esta tercera etapa en todos los casos. En el siguiente apartado se analizan los resultados obtenidos para el RCGA.

Análisis poblacional El objetivo de este análisis es el de obtener el tamaño óptimo de población con el cual el algoritmo obtiene los mejores resultados para cada función. Para llevar a cabo este análisis se ejecutó el RCGA realizando un barrido en el tamaño de población utilizando valores entre 10 y 300 con un salto de 10 individuos. En este caso, se realizaron 54000 ejecuciones del algoritmo. Debido a la gran cantidad de datos generados, en este trabajo únicamente se incluyen los resultados para el tamaño de población óptimo en cada caso.

Como tamaño óptimo poblacional se ha escogido aquel que menor valor de CPEM obtenga en cada función. A la vista de los resultados (véase la tercera columna de la tabla 5.2) no es posible extraer una regla para determinar a priori el tamaño de población óptimo para resolver una determinada función o tipo de función. Los tamaños de población obtenidos toman, en general, valores altos por encima de 50 individuos salvo en alguna excepción, como son las funciones *Penalized 1*, *Rastrigin*, *Rosenbrock*, *Schwefel* y *Step*. Para analizar estas excepciones con mayor detenimiento, en la figura 5.3 se muestra el valor de *CPEM* de estas funciones en los casos en los que el tamaño de la población óptimo es menor que 50 individuos y, por lo tanto, no se corresponde con la tendencia que sigue el algoritmo. Como se ve en esta figura, en las funciones *Penalized 1*, *Rosenbrock*, y *Step* todos los tamaños de población obtienen el mismo nivel de error, como consecuencia de esto, es indiferente cuál de ellos escoger para las ejecuciones. En el caso de las funciones *Rastrigin* y *Schwefel* hasta tamaño de población 90 los resultados de *CPEM* tienen el mismo nivel de error que en los casos de dimensión 10 y 30, por tanto, se podrían utilizar hasta 90 individuos sin empeorar el rendimiento del algoritmo. De este análisis se extrae la conclusión general de que el RCGA necesita tamaños de población altos para resolver las funciones que se le presentan. Este resultado será contrastado en una sección posterior de comparación de algoritmos.

Análisis de resultados

Utilizando los tamaños de población óptimos obtenidos en el apartado anterior pasamos a analizar el comportamiento del algoritmo RCGA siguiendo la metodología descrita en el apartado 4.3.2.

En la tabla 5.2 se muestran los resultados obtenidos por este algoritmo para el conjunto de funciones de prueba. Estos se organizan siguiendo el mismo orden que la tabla 4.1. Las columnas de la tabla 5.2 muestran, de izquierda a derecha, el nombre de cada función, su dimensión, el tamaño de la población que obtuvo el mejor resultado, la tasa de éxito, el valor

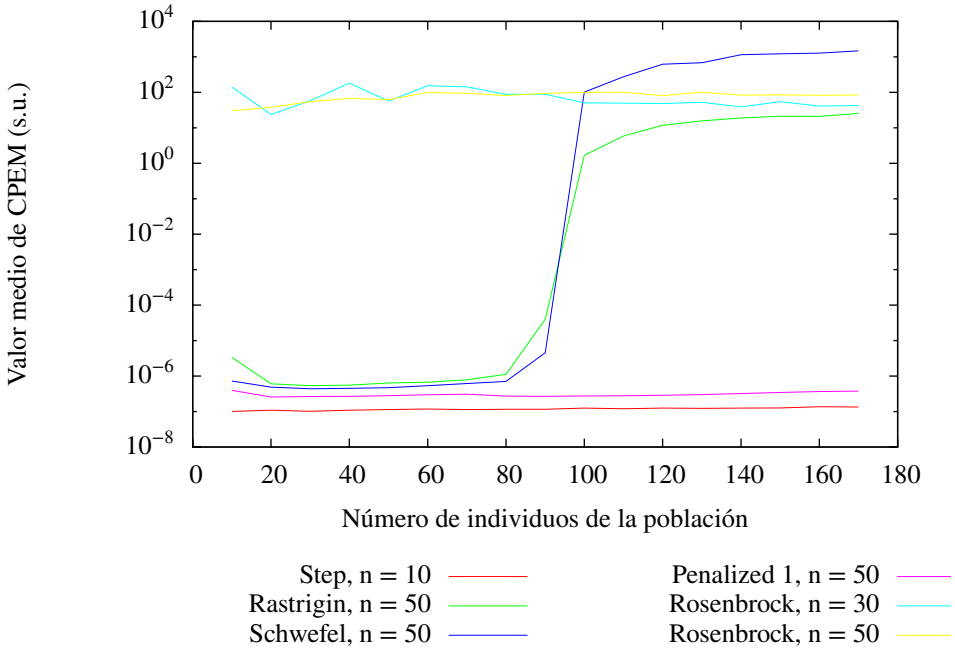


Figura 5.3: Análisis de los casos excepcionales en el análisis poblacional del AG

de *CPEM* y el error genotípico. Como se mencionó en la sección 4.3.1, estos valores se obtuvieron considerando que una ejecución está resuelta cuando el valor de *CPEM* es menor que 10^{-6} y que cuanto menor es el valor de *CPEM*, mayor es la velocidad de convergencia del algoritmo.

Tabla 5.2: Resultados obtenidos por el AG en las funciones del conjunto de prueba

Función	D	N	SR	CPEM	G_{NE}
Axis Parallel	10	110	1.00	$1.55e{-7} \pm 1.73e{-8}$	$1.72e{-6} \pm 4.05e{-7}$
	30	130	1.00	$2.21e{-7} \pm 2.45e{-8}$	$6.97e{-7} \pm 1.33e{-7}$
	50	130	1.00	$4.27e{-7} \pm 4.63e{-8}$	$3.51e{-7} \pm 5.68e{-8}$
Schwefel 2.22	10	100	1.00	$1.66e{-7} \pm 6.73e{-9}$	$1.38e{-8} \pm 3.02e{-9}$
	30	140	1.00	$2.36e{-7} \pm 4.91e{-9}$	$4.67e{-9} \pm 6.18e{-10}$
	50	150	1.00	$6.10e{-7} \pm 7.38e{-8}$	$3.72e{-9} \pm 1.30e{-9}$
Sphere	10	120	1.00	$1.53e{-7} \pm 9.18e{-9}$	$3.20e{-6} \pm 5.81e{-7}$
	30	120	1.00	$1.85e{-7} \pm 1.41e{-8}$	$1.83e{-6} \pm 7.48e{-8}$
	50	120	1.00	$3.45e{-7} \pm 2.79e{-8}$	$1.42e{-6} \pm 1.49e{-7}$
Step	10	90	1.00	$6.29e{-8} \pm 5.08e{-8}$	$3.83e{-3} \pm 9.03e{-4}$
	30	120	1.00	$9.02e{-8} \pm 2.36e{-8}$	$2.86e{-3} \pm 4.97e{-4}$
	50	10	1.00	$1.01e{-7} \pm 2.46e{-8}$	$2.90e{-3} \pm 2.11e{-4}$
SumOf	10	60	1.00	$4.60e{-8} \pm 1.73e{-8}$	$7.22e{-2} \pm 1.58e{-2}$
	30	190	1.00	$3.10e{-8} \pm 5.00e{-9}$	$1.78e{-1} \pm 2.26e{-2}$
	50	70	1.00	$2.62e{-8} \pm 6.72e{-9}$	$2.74e{-1} \pm 2.67e{-2}$
Easom	2	70	1.00	$1.52e{-7} \pm 3.12e{-8}$	$8.46e{-4} \pm 5.52e{-4}$
Schwefel 2.21	10	200	0.00	$2.16e{-2} \pm 4.64e{-2}$	$1.39e{-4} \pm 3.10e{-4}$
	30	300	0.00	$6.29e{-3} \pm 4.29e{-3}$	$3.72e{-5} \pm 2.61e{-5}$
	50	150	0.00	$1.08e{-1} \pm 2.88e{-2}$	$6.42e{-4} \pm 1.78e{-4}$
Colville	4	200	0.24	$4.36e{-1} \pm 8.28e{-1}$	$3.31e{-2} \pm 2.83e{-2}$
Matyas	2	140	1.00	$1.60e{-7} \pm 4.65e{-8}$	$4.24e{-4} \pm 3.32e{-4}$
Perm	10	90	0.00	$7.80e{-2} \pm 1.93e{-1}$	$2.72e{-2} \pm 9.87e{-3}$
	30	220	0.00	$1.38e{+0} \pm 2.31e{+0}$	$4.90e{-3} \pm 2.62e{-3}$
	50	160	0.00	$2.75e{+0} \pm 5.08e{+0}$	$2.47e{-3} \pm 7.65e{-4}$

Tabla 5.2 -- continua desde la página anterior

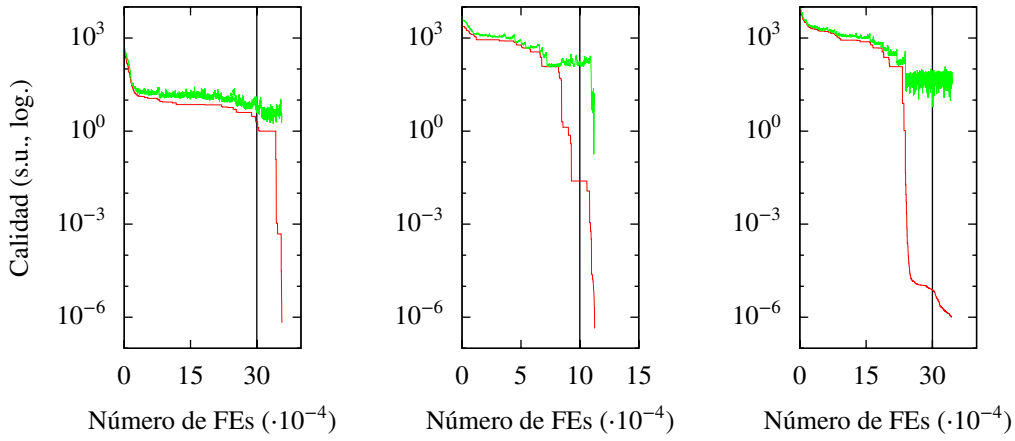
Función	D	N	SR	CPEM	G_{NE}
Schwefel 1.2	10	200	0.00	$2.71e{-1} \pm 3.41e{-1}$	$2.49e{-3} \pm 1.82e{-3}$
	30	170	0.00	$3.99e{+1} \pm 1.92e{+1}$	$2.08e{-2} \pm 5.23e{-3}$
	50	100	0.00	$3.98e{+2} \pm 1.55e{+2}$	$5.04e{-2} \pm 1.05e{-2}$
Zakharov	10	200	0.32	$7.66e{-5} \pm 1.85e{-4}$	$2.18e{-4} \pm 3.00e{-4}$
	30	120	0.00	$5.63e{-1} \pm 4.87e{-1}$	$1.66e{-2} \pm 7.61e{-3}$
	50	70	0.00	$1.56e{+1} \pm 7.94e{+0}$	$7.25e{-2} \pm 1.74e{-2}$
Aluffi-Pentini's	2	60	1.00	$6.64e{-8} \pm 1.07e{-8}$	$1.35e{-2} \pm 8.23e{-3}$
Becker and Lago	2	60	1.00	$7.51e{-8} \pm 1.10e{-8}$	$6.64e{-1} \pm 3.21e{-1}$
Bohachevsky 1	2	60	1.00	$9.62e{-8} \pm 1.41e{-8}$	$3.43e{-4} \pm 1.79e{-4}$
Cosine Mixture	10	90	1.00	$9.28e{-8} \pm 1.56e{-8}$	$8.70e{-5} \pm 9.91e{-6}$
	30	110	1.00	$1.28e{-7} \pm 1.42e{-8}$	$4.93e{-5} \pm 2.94e{-6}$
	50	110	1.00	$2.04e{-7} \pm 1.48e{-8}$	$3.83e{-5} \pm 4.14e{-6}$
Rastrigin	10	110	1.00	$6.15e{-7} \pm 1.86e{-7}$	$1.62e{-5} \pm 2.71e{-5}$
	30	100	0.72	$6.29e{-6} \pm 1.61e{-5}$	$6.57e{-6} \pm 8.80e{-6}$
	50	30	1.00	$5.38e{-7} \pm 6.42e{-8}$	$1.98e{-6} \pm 2.73e{-7}$
Schwefel	10	120	0.96	$5.40e{-2} \pm 2.65e{-1}$	$8.55e{-5} \pm 4.13e{-4}$
	30	100	0.92	$7.06e{-5} \pm 3.29e{-4}$	$3.83e{-6} \pm 8.11e{-6}$
	50	30	1.00	$4.39e{-7} \pm 6.57e{-8}$	$1.81e{-6} \pm 3.48e{-7}$
Ackleys	10	110	1.00	$2.21e{-7} \pm 1.36e{-8}$	$8.00e{-9} \pm 1.19e{-9}$
	30	130	1.00	$2.93e{-7} \pm 5.50e{-8}$	$7.83e{-9} \pm 5.41e{-10}$
	50	140	0.96	$7.67e{-7} \pm 1.44e{-7}$	$7.89e{-9} \pm 5.21e{-10}$
Griewank	30	300	1.00	$4.05e{-7} \pm 9.47e{-9}$	$1.71e{-6} \pm 1.93e{-7}$
	50	120	0.88	$4.50e{-3} \pm 1.35e{-2}$	$3.39e{-4} \pm 9.56e{-4}$
Levy	10	90	1.00	$9.54e{-8} \pm 2.34e{-8}$	$1.21e{-4} \pm 1.95e{-5}$
	30	120	1.00	$1.41e{-7} \pm 1.15e{-8}$	$7.21e{-5} \pm 3.96e{-6}$
	50	110	1.00	$2.18e{-7} \pm 1.46e{-8}$	$5.53e{-5} \pm 3.89e{-6}$
Penalized 1	10	100	1.00	$1.34e{-7} \pm 3.82e{-8}$	$4.23e{-5} \pm 9.02e{-6}$
	30	130	1.00	$1.87e{-7} \pm 2.14e{-8}$	$4.37e{-5} \pm 3.53e{-6}$
	50	20	1.00	$2.58e{-7} \pm 6.56e{-8}$	$4.37e{-5} \pm 8.22e{-6}$
Penalized 2	10	170	1.00	$2.18e{-7} \pm 9.73e{-8}$	$1.88e{-5} \pm 3.87e{-6}$
	30	140	1.00	$2.42e{-7} \pm 7.67e{-8}$	$1.03e{-5} \pm 2.88e{-6}$
	50	100	1.00	$3.39e{-7} \pm 2.67e{-8}$	$9.36e{-6} \pm 4.27e{-6}$
Beale	2	150	1.00	$2.30e{-7} \pm 7.47e{-8}$	$3.15e{-2} \pm 1.54e{-2}$
Bohachevsky 2	2	60	1.00	$9.85e{-8} \pm 2.06e{-8}$	$7.69e{-4} \pm 1.42e{-3}$
Dekkers and Aarts	2	50	1.00	$4.38e{-8} \pm 1.00e{-8}$	$7.01e{-1} \pm 4.88e{-1}$
Goldstein Price	2	80	1.00	$1.13e{-7} \pm 1.72e{-8}$	$2.18e{-3} \pm 1.37e{-3}$
Griewank	10	180	0.48	$1.03e{-2} \pm 1.25e{-2}$	$2.36e{-3} \pm 2.48e{-3}$
Hartman 3	3	50	1.00	$3.77e{-8} \pm 6.63e{-9}$	$8.48e{-2} \pm 4.02e{-2}$
Hartman 6	6	120	0.92	$1.01e{-2} \pm 3.43e{-2}$	$3.38e{-1} \pm 4.31e{-1}$
Rosenbrock	10	190	0.00	$5.28e{+0} \pm 2.51e{+0}$	$3.25e{-2} \pm 2.25e{-2}$
	30	20	0.00	$2.38e{+1} \pm 1.84e{+1}$	$9.52e{-2} \pm 6.81e{-2}$
	50	10	0.00	$3.03e{+1} \pm 3.47e{+1}$	$6.71e{-2} \pm 5.43e{-2}$
Kowalik's	4	180	0.00	$5.13e{-4} \pm 1.44e{-4}$	$3.98e{-1} \pm 1.59e{-1}$
Shekel Family 5	4	110	0.72	$2.09e{+0} \pm 3.35e{+0}$	$2.40e{-1} \pm 2.30e{-1}$
Shekel Family 7	4	180	0.84	$9.66e{-1} \pm 2.29e{+0}$	$1.90e{-2} \pm 5.53e{-2}$
Shekel Family 10	4	150	0.96	$2.68e{-1} \pm 1.31e{+0}$	$8.10e{-3} \pm 3.97e{-2}$
Shekel's Foxholes	2	60	0.88	$1.19e{-1} \pm 4.28e{-1}$	$7.46e{-2} \pm 1.00e{-1}$
SixHump Camel Back	2	50	1.00	$5.57e{-8} \pm 8.02e{-9}$	$1.10e{-1} \pm 9.73e{-2}$

Configuración básica En primer lugar, se analiza el RCGA utilizando la configuración propuesta por [Herrera et al., 1998]. Teniendo en cuenta los resultados de la tabla 5.2 y las características de separabilidad y modalidad de las funciones, queda claro que la separabi-

lidad es la característica que afecta en mayor medida al comportamiento del RCGA. Por un lado, el algoritmo resuelve todas las funciones *L-Separables* y *NL-Separables*, excepto la función *Schwefel 2.21*. Por el contrario, presenta problemas para resolver las funciones *No-Separables*. El RCGA obtiene una tasa de éxito del 100% en 7 funciones de 8 de muy baja dimensionalidad $D = 2$ y $D = 3$, siendo D la dimensión del problema. Los resultados empeoran a medida que crecen las dimensiones. Estos resultados generales serán analizados con mayor detalle en los siguientes párrafos.

En las funciones *Rastrigin* y *Schwefel*, ambas de tipo *L-Separables*, la tasa de éxito es menor que el 100%, lo cual indica que hay algunas ejecuciones que no se resuelven. Concretamente, la función *Rastrigin* para dimensión 30 tiene un SR del 72% y la función *Schwefel* para dimensiones 10 y 30 obtiene un SR del 96% y el 92%, respectivamente. Para analizar el caso de la función *Rastrigin* se debe considerar el valor del error genotípico. Como se comentó en la sección 4.2, no se pueden obtener conclusiones acerca del comportamiento del algoritmo analizando únicamente el error genotípico, pero en este caso se puede utilizar comparándolo con el resultado obtenido para dicha función con otras dimensiones. El RCGA obtiene un error genotípico de $1.62e{-}5$ para dimensión 10 y $1.98e-6$ para dimensión 50, mientras que en el caso problemático (dimensión 30) el valor obtenido es $6.57e-6$. Este valor indica que la solución está cerca del óptimo. De hecho, la razón principal de los fallos del RCGA en esta función es que el algoritmo necesita más generaciones para resolverla. En la figura 5.4a se muestra como permitiendo más evaluaciones el algoritmo alcanza el valor de calidad considerado óptimo. En esta figura la línea negra marca el límite de evaluaciones permitido en este procedimiento experimental para que la posterior comparación de algoritmos sea justa. Lo mismo ocurre en la función *Schwefel* para dimensiones 10 y 30. Comparando el error genotípico obtenido en estas dimensiones ($8.55e{-}5$ y $3.83e{-}6$) con el error genotípico para dimensión 50 ($1.81e{-}6$), donde el SR es de un 100%, los valores obtenidos indican que las soluciones están cerca del óptimo. En las figuras 5.4b y 5.4c se muestra el comportamiento del RCGA ante la función *Schwefel* permitiendo al algoritmo un número mayor de generaciones, donde de nuevo se confirma que el criterio de parada utilizado perjudica al RCGA, aunque con un mayor número de generaciones, la respuesta del algoritmo en funciones de tipo *L-Separables* sería totalmente satisfactoria.

A partir de la información obtenida utilizando el algoritmo de multimodalidad presentado en la sección 4.1.3 que se muestra en la tabla 4.4, se puede observar que el centro de atracción más amplio de la función *Schwefel* no es el óptimo, lo cual explica por qué esta función es más difícil que otras funciones *L-Separables*. Esta característica provoca que esta función sea más difícil de resolver, no solamente para un RCGA(comprobaremos en análisis posteriores que esta característica afecta a otros AEs). Existen más funciones en el conjunto de prueba con esta característica que serán comentadas más adelante. Para comprender claramente el comportamiento de un algoritmo en funciones *L-Separables*, es necesario analizar profundamente sus operadores, cómo funcionan y la influencia de los parámetros. Está demostrado que la optimización de cada parámetro en procesos independientes es la mejor estrategia para resolver las funciones *L-Separables* [Sutton et al., 2007]. Para seguir esta estrategia, los algoritmos deben modificar únicamente un parámetro del genotipo en cada generación. A este tipo de movimiento sobre el espacio de calidad se le denomina *búsqueda ortogonal* debido a que la trayectoria que sigue el genotipo en el espacio de calidad es siempre ortogonal a uno de los ejes de coordenadas. Obviamente, si se modifica más de un gen o parámetro del genotipo en cada generación se pierde esta ortogonalidad. Se puede establecer un nivel de ortogonalidad en la búsqueda dependiendo del número de genes que se modifican en cada generación. El RCGA es capaz de controlar este nivel de ortogonalidad utilizando los operadores de cruce



(a) Función Rastrigin $n = 10$ (b) Función Schwefel $n = 10$ (c) Función Schwefel $n = 30$

Figura 5.4: Evoluciones del AG sin límite de evaluaciones para las funciones *Rastrigin* y *Schwefel*. La línea roja representa la evolución de la calidad del mejor individuo y la línea verde la evolución de la calidad media de la población. La línea vertical negra representa el límite de evaluaciones permitidas en las pruebas de este trabajo.

y mutación. Una probabilidad de mutación de un 5% y una probabilidad de cruce del 60%, utilizadas en las pruebas de este trabajo, provocan que la probabilidad de no alterar el valor de un gen sea de un 39.8% ($(1 - 0.60) \times (1 - 0.005)$), y esto favorece al RCGA a la hora de resolver funciones *L-Separables* como muestra los datos de la tabla de resultados.

En el caso de las funciones *NL-Separables*, el RCGA no es capaz de resolver algunas ejecuciones de las funciones *Ackleys* y *Griewank* para dimensión 50, con una SR del 96% y el 88% respectivamente. La función *Ackleys* no representa ningún problema, ya que su error genotípico en dimensión 50 ($7.89e{-}9$), es similar al error genotípico de dimensión 10 y 30 ($7.93e{-}9$ y $8.00e{-}9$) donde el SR es del 100%. De hecho, la ejecución de la función *Ackleys* que se considera no resuelta obtiene un *CPEM* de $1.31e{-}6$, lo cual es casi un éxito. Teniendo en cuenta la modalidad y la información de la tabla 4.4, la función *Ackleys* es la que mayor número de centros de atracción presenta (65^n). Este resultado puede hacer pensar que dicha función es la de mayor dificultad, sin embargo, tras los resultados obtenidos se puede concluir que el número de centros de atracción no es la propiedad que define la dificultad de una función. En esta función, a pesar del elevado número de centros de atracción, la distribución de los mismos (véase figura 4.20) facilita el proceso de búsqueda. La función *Griewank* tiene el mismo problema que ya fue comentado anteriormente la función *Schwefel*: el centro de atracción más amplio no es el óptimo. Esto provoca que el RCGA converja con mayor facilidad en óptimos locales que otras funciones que no presentan esta característica.

Donde realmente falla el RCGA, centrándonos en las funciones *NL-Separables*, es en las funciones *Schwefel 2.21* y *Griewank*. Para analizar estos casos tendremos en cuenta la modalidad de la funciones y la información extraída de la aplicación de los algoritmos de modalidad de la sección 4.1.3 que se presenta en las tablas 4.4 y 4.5. La función *Schwefel 2.21* es unimodal y como puede verse en la tabla 4.5 la longitud del camino hasta el óptimo es muy larga. Esta característica degrada el rendimiento del RCGA debido al modo en que este algoritmo ajusta el paso de mutación. Como veremos más adelante, esta característica no afecta solamente al RCGA sino que también afecta a otros AEs, aunque no siempre en la misma medida ya que cada AE tiene su modo de ajustar el paso de mutación. La función

Griewank es multimodal y, observando la información obtenida tras el análisis de multimodalidad, el centro de atracción del óptimo global no es el que tiene el mayor tamaño. Esto representa un problema para el RCGA ya que el proceso de búsqueda tiende hacia el centro de atracción de mayor tamaño y no hacia el óptimo. Como resultado, un 12% de las ejecuciones en el caso de dimensión 50 convergen hacia óptimos locales. A pesar de estos dos casos, el comportamiento general del RCGA en las funciones *NL-Separables* es exitoso, lo cual es lógico considerando que el proceso de búsqueda en este tipo de funciones es muy similar al que debe seguirse en las funciones *L-Separables*. Las dependencias entre parámetros determinan solamente la forma del espacio de calidad, pero no el valor del punto óptimo. En consecuencia, durante el proceso de optimización, el algoritmo debe buscar el mismo valor para una determinada variable, independientemente del valor de otros genes. De nuevo, este proceso se realiza mejor buscando en direcciones ortogonales, es decir, modificando solo algunos parámetros del cromosoma en cada generación.

Las funciones *No-Separables* son las que mayores dificultades presentan para el RCGA debido a las fuertes dependencias entre sus parámetros. Como se muestra en la tabla 5.2, el RCGA obtiene resultados pobres en estas funciones, resolviendo con una tasa de éxito del 100% solamente 7 de 19 funciones (6 de 2 dimensiones y 1 función de 3 dimensiones). Los resultados se degradan a medida que crece el número de parámetros de las funciones. Concretamente, resuelve 1 función de 2 dimensiones con un SR del 88% (*Shekel's Foxholes*), 3 funciones de dimensión 4 (*Shekel Family 5* con un 72%, *Shekel Family 7* con un 84%, *Shekel Family 10* con un 96%) y 1 función de dimensión 6 (*Hartman 6* con un 92%). En el resto de los casos las tasas de éxito se encuentran entre el 0% (no resuelve ninguna ejecución) y un 48% (resuelve la mitad de las pruebas ejecutadas). Estas funciones son: 1 función de dimensión 4 (*Colville*) y todas las funciones de dimensión mayor que 10 (*Perm, Schwefel, Zakharov, Griewank y Ronsenbrock*). Además de la no separabilidad, para analizar las razones de estos resultados es necesario tener en cuenta la modalidad del espacio de calidad. En el caso de las funciones unimodales debemos fijarnos en la longitud del camino hasta el óptimo (tabla 4.5), como se hizo anteriormente con la función *Schwefel 2.21*. Según los datos de esta tabla las funciones *Perm*, *Schwefel 1.2* y *Zakharov* (de alta dimensionalidad) y la función *Colville* (de dimensión cuatro) tienen los caminos hasta el óptimo más largos. Como ya se comentó en el caso de la función *Schwefel 2.21*, el ajuste del paso de mutación hace que el RCGA sea lento a la hora de resolver estas funciones y que en muchos casos no llegue a resolverlas.

Siguiendo con el análisis de modalidad, teniendo en cuenta las funciones multimodales de baja dimensionalidad y los resultado la tabla 4.4, las funciones *No-Separables* se pueden dividir en dos grupos: aquellas donde el centro de atracción óptimo no es el de mayor tamaño y aquellas en las que sí. En el primer grupo se incluyen las funciones *Griewank* de dimensión 10, *Hartman 6* de dimensión 6, *Shekel Family 5* y *Kowaliks* de dimensión 4. Las tasas de éxito de estas funciones son 48%, 92%, 72% y 0%, respectivamente. En el segundo grupo de funciones, aquellas en las que el centro de atracción óptimo sí es el de mayor tamaño, se incluyen las funciones *Shekel Family 7* (dimensión 4, SR 84%), *Shekel Family 10* (dimensión 4, SR 96%) y *Shekel's Foxholes* (dimensión 2, SR 88%). En ambos casos, teniendo en cuenta los valores obtenidos de *CPEM* y error genotípico además de la tasa de éxito, las ejecuciones no resueltas convergen hacia los óptimos locales. El rendimiento empeora en menor medida en el segundo grupo de funciones, debido a que éstas no son deceptivas, es decir el centro de atracción de mayor tamaño es el centro de atracción del óptimo y el proceso de búsqueda tiende hacia él. Por lo tanto, en general, estos valores en cuanto al SR indican que este algoritmo no debería ser recomendado cuando se trabaje con funciones *No-Separables*, especialmente cuando estas son multimodales.

Resumiendo, en general, los malos resultados indican que este algoritmo no debe ser recomendado en el caso de tratar de resolver una función no separable. Más concretamente, el algoritmo tampoco es aconsejable si además de tratarse de una función no separable esta es unimodal con un camino largo hasta el óptimo o multimodal con el centro de atracción más amplio diferente al del óptimo [1]. La principal razón detrás de estos resultados es que, para resolver funciones no separables, un algoritmo debe aprender las dependencias entre parámetros, lo que no ocurre en el caso de los operadores del RCGA ya que no utiliza información de otros genes del cromosoma para generar nuevos valores.

Resumiendo, los resultado obtenidos por el RCGA utilizando su configuración básica permiten realizar la siguiente caracterización del mismo:

- Su rendimiento es satisfactorio en las funciones *L-Separables* y *NL-Separables* debido a su capacidad para buscar en direcciones ortogonales.
- Presenta problemas en las funciones *No-Separables* porque no tiene en cuenta las dependencias entre genes durante el proceso de optimización. El rendimiento empeora a medida que crecen las dimensiones: cuando las dimensiones del problema son 10 o más el RCGA nunca resuelve las funciones del conjunto de prueba.
- En funciones multimodales, la presencia de centros de atracción más amplios que el óptimo o un número elevado de centros de atracción reducen las probabilidades de encontrar el óptimo. Esto ocurre tanto en funciones *L/NL-Separables* como en funciones *No-Separables*. En las primeras, ocurre solamente en algunas ejecuciones y cuando la dimensionalidad es baja. En las segundas, el RCGA presenta un mayor número de fallos y además también se dan en funciones de baja dimensionalidad.
- En funciones unimodales, un camino hasta el óptimo de longitud elevada provoca que el rendimiento del RCGA empeore sin importar la separabilidad. Esto es debido a que la estrategia de búsqueda del RCGA no permite un buen ajuste dinámico del paso de mutación durante la optimización.

Variaciones sobre la configuración básica En la sección anterior se han explicado los resultados obtenidos por el RCGA utilizando los parámetros recomendados en [Herrera et al., 1998] cuyo resultado está contrastado en este tipo de problemas. Sin embargo, como acabamos de observar, existen funciones dentro del conjunto de prueba utilizado que con dicha configuración no se resuelven. En esta sección se analizaran estas funciones y, utilizando las conclusiones obtenidas en la sección anterior sobre el comportamiento de este algoritmo, se intentará ajustar los parámetros del algoritmo para tratar de mejorar los resultados obtenidos.

En primer lugar se identificarán las funciones no resueltas y se analizarán sus características. Concretamente, el RCGA utilizado en este trabajo no resuelve las siguientes funciones:

- En cuanto a funciones unimodales, no resuelve las funciones *Perm*, *Schwefel 2.21*, *Colville*, *Schwefel 1.2* y *Zakharov*. Todas son *No-Separables* excepto la función *Schwefel 2.21*. En la clasificación de longitud de camino al óptimo, las cuatro primeras funciones ocupan los primeros puestos de la misma. La función *Zakharov* ocupa el séptimo lugar de trece funciones analizadas y su longitud de camino, 220.41, está por debajo de la media de las funciones analizadas (siendo la media 315.37).

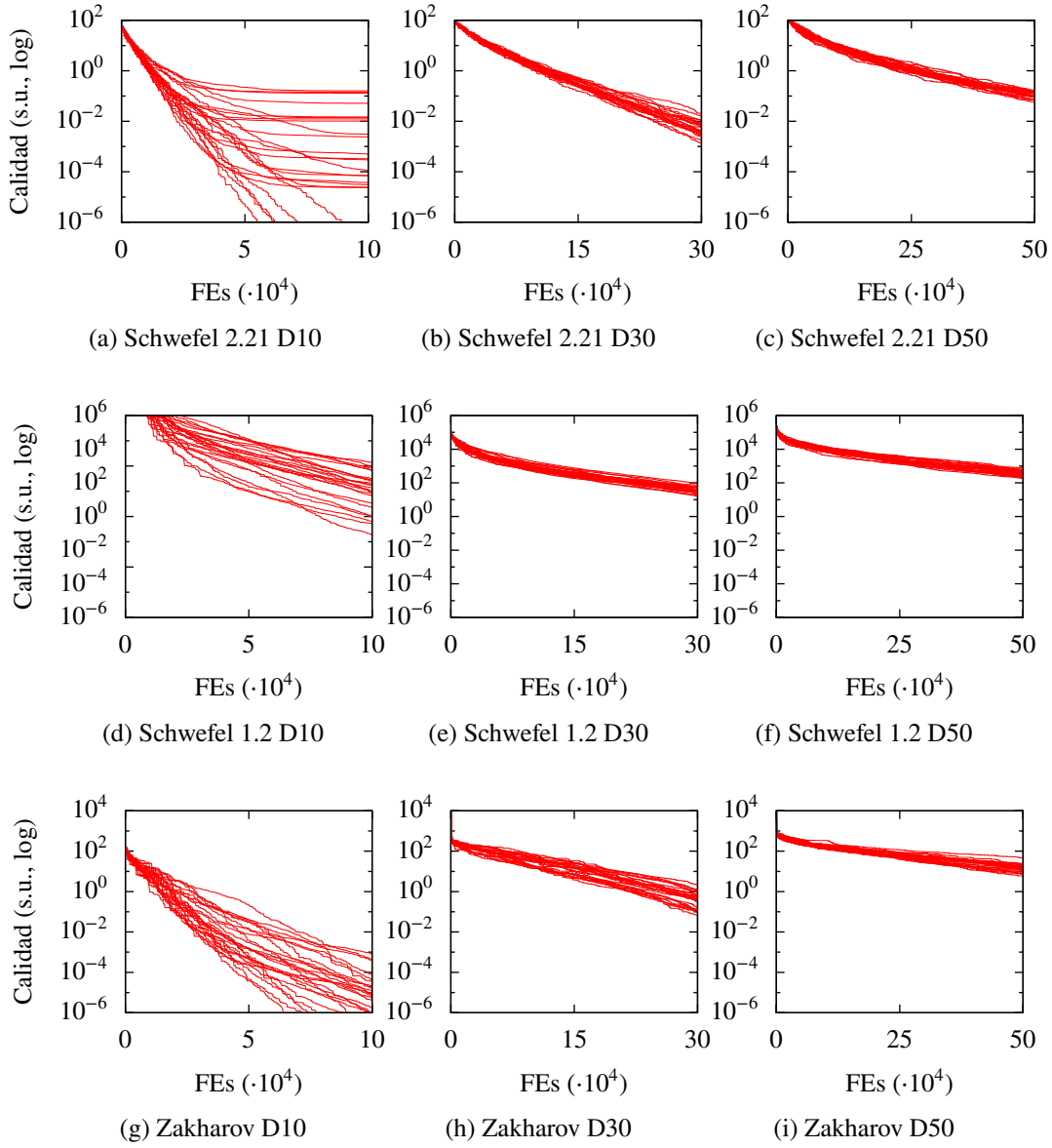
[1] Esta recomendación es válida. al menos, utilizando los operadores y la configuración indicada en [Herrera et al., 1998].

Función	D	N	FEs	SR	CPEM	G_{NE}
Schwefel 2.21	10	200	$10000 \cdot n$	0.20	$2.16e{-2} \pm 4.64e{-2}$	$1.39e{-4} \pm 3.10e{-4}$
			$100000 \cdot n$	0.48	$3.07e{-3} \pm 5.51e{-3}$	$1.70e{-5} \pm 3.10e{-5}$
	30	300	$10000 \cdot n$	0.00	$6.29e{-3} \pm 4.29e{-3}$	$3.72e{-5} \pm 2.61e{-5}$
			$100000 \cdot n$	1.00	$1.10e{-7} \pm 1.87e{-8}$	$5.69e{-9} \pm 5.89e{-10}$
	50	150	$10000 \cdot n$	0.00	$1.08e{-1} \pm 2.88e{-2}$	$6.42e{-4} \pm 1.78e{-4}$
			$100000 \cdot n$	0.80	$9.43e{-7} \pm 3.08e{-7}$	$6.35e{-9} \pm 1.40e{-9}$
Schwefel 1.2	10	200	$10000 \cdot n$	0.00	$2.71e{-1} \pm 3.41e{-1}$	$2.49e{-3} \pm 1.82e{-3}$
			$100000 \cdot n$	1.00	$3.16e{-7} \pm 5.17e{-8}$	$5.77e{-6} \pm 3.03e{-7}$
	30	170	$10000 \cdot n$	0.00	$3.99e{+1} \pm 1.92e{+1}$	$2.08e{-2} \pm 5.23e{-3}$
			$100000 \cdot n$	1.00	$4.19e{-8} \pm 3.59e{-8}$	$3.46e{-6} \pm 8.24e{-8}$
	50	100	$10000 \cdot n$	0.00	$3.98e{+2} \pm 1.55e{+2}$	$5.04e{-2} \pm 1.05e{-2}$
			$100000 \cdot n$	0.00	$1.43e{-4} \pm 7.31e{-5}$	$3.09e{-5} \pm 8.98e{-6}$
Zakharov	10	200	$10000 \cdot n$	0.32	$7.66e{-5} \pm 1.85e{-4}$	$2.18e{-4} \pm 3.00e{-4}$
			$100000 \cdot n$	1.00	$1.21e{-7} \pm 5.12e{-8}$	$4.06e{-5} \pm 2.03e{-6}$
	30	120	$10000 \cdot n$	0.00	$5.63e{-1} \pm 4.87e{-1}$	$1.66e{-2} \pm 7.61e{-3}$
			$100000 \cdot n$	1.00	$3.83e{-7} \pm 5.45e{-8}$	$2.42e{-5} \pm 1.42e{-7}$
	50	70	$10000 \cdot n$	0.00	$1.56e{+1} \pm 7.94e{+0}$	$7.25e{-2} \pm 1.74e{-2}$
			$100000 \cdot n$	0.00	$1.08e{-3} \pm 1.34e{-3}$	$5.47e{-4} \pm 2.89e{-4}$

Tabla 5.3: Comparación de los resultados del GA en las funciones *Schwefel 2.21*, *Schwefel 1.2* y *Zakharov* incrementando el número de FEs permitidas de $10000 \cdot n$ a $100000 \cdot n$.

- Por otro lado, las funciones *Kowaliks*, *Griewank*, *Shekel Family 5* y *Rosenbrock* son funciones multimodales, todas ellas *No-Separables*. En las tres primeras, el centro de atracción óptimo no es el de mayor tamaño. No ocurre lo mismo en la función *Rosenbrock*, esta función presenta nueve centros de atracción, dos de ellos son los principales y ambos presentan tamaños similares, aunque el óptimo es de mayor tamaño.

Como ya ha sido mencionado en el desarrollo de este trabajo, las funciones unimodales presenta un único óptimo global, y por tanto, es posible alcanzarlo desde cualquier punto del espacio de calidad. Dado el caso de que un AE no resuelva una función unimodal, esto puede deberse a dos motivos: la velocidad de convergencia del algoritmo es lenta o la población converge hacia una solución no óptima por falta de información de gradiente en el espacio de calidad. Para detectar cuál de los dos problemas anteriores ocurre durante las ejecuciones realizadas en necesario examinar las gráficas de evolución. Las gráficas que se presentan en las figuras 5.5 y 5.6 representan las evoluciones ejecutadas para las funciones unimodales que presentan problemas al ser resueltas mediante el RCGA utilizado. Como ya se ha visto en la sección anterior, el RCGA utilizado en este trabajo presenta una velocidad de convergencia lenta. Lo cual representa un inconveniente al trabajar con un test de parada que limita el número de evaluaciones, y como consecuencia, el número de generaciones ejecutadas. Uno de los motivos de esta lentitud es su requisito poblacional, ya que, generalmente este algoritmo necesita más individuos que otros AEs. Ya ha sido demostrado como ciertas ejecuciones de las funciones *Rastrigin* o *Schwefel* no se resuelven en el número de FEs establecidas, pero que pueden ser resueltas si se permiten algunas iteraciones más del algoritmo. Estudiando las gráficas de evolución de las funciones unimodales (ver figuras 5.5 y 5.6) que el RCGA no resuelve, se puede concluir que esto mismo ocurre en las funciones *Schwefel 2.21*, *Schwefel 1.2* y *Zakharov*. La velocidad de convergencia de estas funciones es lenta y, una vez alcanzado el límite de FEs permitido, la población no ha convergido en una solución no óptima. Aumentando el límite de FEs permitidos hasta $100000 \cdot n$ evaluaciones, siendo n la dimensión del problema, obtenemos los resultados que se presentan en la tabla 5.3.

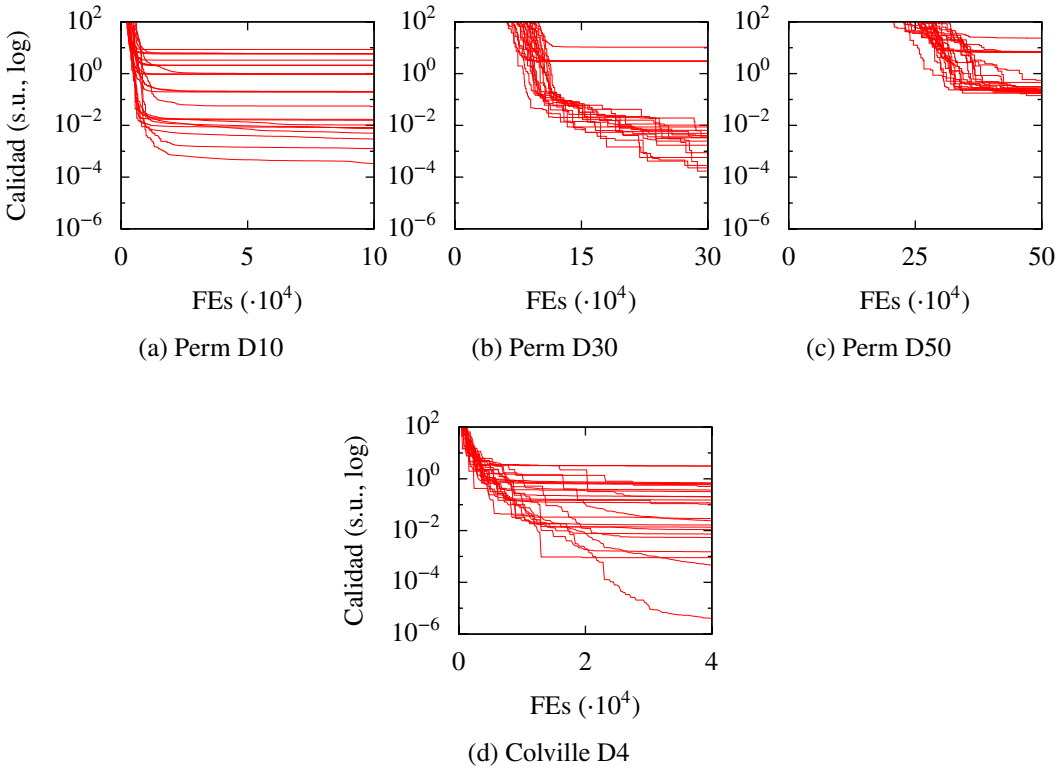


(a) Schwefel 2.21 D10 (b) Schwefel 2.21 D30 (c) Schwefel 2.21 D50

(d) Schwefel 1.2 D10 (e) Schwefel 1.2 D30 (f) Schwefel 1.2 D50

(g) Zakharov D10 (h) Zakharov D30 (i) Zakharov D50

Figura 5.5: Ejecuciones realizadas para las funciones unimodales *Schwefel 2.21*, *Schwefel 1.2* y *Zakharov*. El eje x representa el número de FEs ejecutadas. El eje y representa el valor de calidad del mejor individuo de la población en escala logarítmica.



(a) Perm D10 (b) Perm D30 (c) Perm D50

(d) Colville D4

Figura 5.6: Ejecuciones realizadas para las funciones unimodales *Perm* y *Colville*. El eje x representa el número de FEs ejecutadas. El eje y representa el valor de calidad del mejor individuo de la población en escala logarítmica.

Como muestran los resultados de dichas tablas, la hipótesis planteada a partir de la observación de las gráficas de evolución de la calidad era cierta y, suavizando el test de parada, permitiendo un número mayor de FEs, los resultados de todas las ejecuciones mejoran. Únicamente destacar cuatro casos, la función *Schwefel 2.21* para dimensión 10 y todas las funciones para dimensión 50. En el primero de ellos (*Schwefel 2.21* D10), a pesar de la mejora, en las ejecuciones realizadas para dimensión 10 continúan existiendo algunas que no se resuelven. Este comportamiento será analizado más adelante con el resto de funciones unimodales. En dimensión 50, las tres funciones analizadas mejoran su valor de CPEM aunque ninguna obtiene un SR del 100%. Si se analizan los valores de G_{NE} y se comparan con los obtenidos para dimensión 30, donde sí se resuelven todas las ejecuciones, se observa que los valores obtenidos son muy cercanos. Por lo tanto, es posible concluir que aumentando el valor de FEs el algoritmo resolvería todas las ejecuciones realizadas.

Resumiendo, en el caso de las funciones *Schwefel 2.21*, *Schwefel 1.2* y *Zakharov*, la velocidad de convergencia lenta del RCGA no permite que estas funciones sean resueltas en el tiempo establecido.

De las funciones unimodales presentadas inicialmente, la función *Colville* y la función *Perm* no presentaban este problema. Además, en el caso de la función *Schwefel 2.21* hay ejecuciones que no se resuelven aunque se incremente el número de FEs. El problema en estos casos es que la población converge demasiado rápido debido a la falta de información de gradiente en el espacio de calidad. Son funciones que presentan amplias planicies y esta característica es muy perjudicial para el rendimiento del RCGA. De estas tres funciones, la función *Schwefel 2.21* es *NL-Separable* y las funciones *Colville* y *Perm* son *No-Separables*.

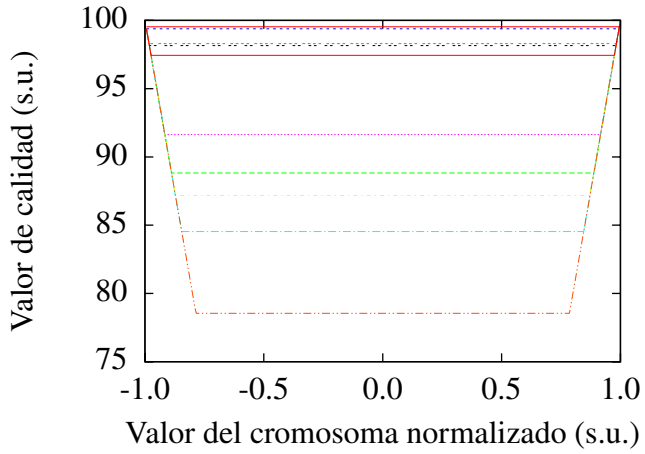


Figura 5.7: Análisis de separabilidad: Función *Schwefel 2.21*

D	N	p_{cr}	SR	CPEM	G_{NE}
10	200	60%	0.48	$3.07e{-3} \pm 5.51e{-3}$	$1.70e{-5} \pm 3.10e{-5}$
		80%	1.00	$6.24e{-7} \pm 5.62e{-8}$	$5.61e{-9} \pm 8.11e{-10}$

Tabla 5.4: Comparación de los resultados del GA para la función *Schwefel 2.21* cuando se incrementa la probabilidad de cruce de un 60% a un 80%.

Como se demostró en el apartado anterior, el RCGA no presenta problemas destacables a la hora de resolver las funciones *NL-Separables*, por esto, en principio, la función *Schwefel 2.21* no debería representar ningún problema. Sin embargo, las dependencias entre variables de esta función hacen que, dependiendo de los valores que tomen las variables, aparezcan superficies planas sin información de gradiente, como se observa en la gráfica de separabilidad de la figura 5.7. En este caso particular, la búsqueda en direcciones ortogonales que favorece a las funciones *NL-Separables* es contraproducente. Por este motivo, para evitar este tipo de búsqueda y favorecer la búsqueda en direcciones diagonales se debería aumentar la probabilidad de cruce del RCGA. Los resultados que se muestran en la tabla 5.4 se han obtenido tras aumentar dicha probabilidad del 60% al 80%. Se observa en dicha tabla que ahora el SR obtenido para la función *Schwefel 2.21* es del 100%.

Teóricamente, un aumento en la probabilidad de cruce debería favorecer también a las funciones *Colville* y *Perm* que presentan las misma topología con planicies que la función *Schwefel 2.21*. Sin embargo, esto no ocurre. Ambas funciones son *No-Separables* y además de un proceso de búsqueda en direcciones diagonales sería necesario un mecanismo que fuese capaz de aprender las dependencias entre variables para obtener resultados mejores a los obtenidos con la configuración original. El RCGA utilizado en este trabajo no tiene este tipo de mecanismo y los operadores modifican cada gen con independencia de las modificaciones realizadas a otros genes del cromosoma.

Las funciones *Kowaliks*, *Griewank*, *Rosenbrock* y *Shekel Familiy 5* son multimodales y *No-Separables*. Al ser no separables, la primera hipótesis que se plantea es que aumentando la probabilidad de cruce y mutación los resultados de estas funciones mejorarán ya que, durante el proceso de búsqueda, se tenderá hacia direcciones diagonales y no ortogonales, lo cual favorece a los algoritmos en funciones *No-Separables*. Además, aumentando estas probabilidades, el nivel de exploración también aumenta, lo cual favorece a los algoritmos en funciones multimodales. Sin embargo, debido a los distintos tipos de multimodalidad que

D	N	p_{cr}	SR	CPEM	G_{NE}
10	180	60%	0.48	$1.02e{-}2 \pm 1.25e{-}2$	$2.36e{-}3 \pm 2.48e{-}3$
		80%	1.00	$1.97e{-}7 \pm 3.68e{-}8$	$9.17e{-}5 \pm 1.35e{-}5$

Tabla 5.5: Comparación de los resultados del GA para la función *Griewank* cuando se incrementa la probabilidad de cruce de un 60% a un 80%.

presentan estas funciones, estos cambios de configuración solamente favorecen a la función *Griewank*. Los resultados obtenidos por dicha función tras aumentar la probabilidad de cruce de un 60% a un 80% se muestran en la tabla 5.5. Como se muestra en la tabla, aumentando la probabilidad de cruce y, como consecuencia, el nivel de exploración, el algoritmo resuelve ahora todas las ejecuciones de la función.

No ocurre lo mismo en las tres funciones restantes (función *Kowaliks*, *Rosenbrock*, *Shekel Family 5*), donde el nivel de no separabilidad es mayor que en la función *Griewank*. Además, son funciones no simétricas en cuanto a modalidad, al contrario que la función *Griewank* donde los centros de atracción se distribuyen simétricamente en el espacio de calidad. En estos tres casos, la población converge muy rápidamente hacia un óptimo local ya que sus centros de atracción son de mayor tamaño que el óptimo o, como ocurre en la función *Rosenbrock*, no siendo el mayor, es de tamaño considerable comparativamente. Al aumentar la probabilidad de cruce, aumentando así el nivel de exploración, mejora la variabilidad media de la población pero el algoritmo continua convergiendo hacia óptimos locales. En estos casos también se ejecutó el algoritmo aumentando el paso de mutación, disminuyendo el parámetro *b* del valor por defecto 5 a un valor de 1. Ejecutando el RCGA con esta configuración sobre las tres funciones anteriores, la población no converge pero continua sin resolverlas. Lo mismo ocurre si se disminuye el paso de mutación incrementando el valor del parámetro *b* de un valor de 5 a un valor de 7. Con este análisis se llega a la conclusión de que la no separabilidad de estas funciones provoca que el RCGA utilizado no las resuelva ya que, como se ha repetido en este trabajo, no aprende dependencias entre variables, clave para el éxito de los algoritmos en este tipo de funciones.

En esta sección, el procedimiento de caracterización desarrollado en este trabajo ha sido aplicado a un algoritmo genético con operadores de codificación real. En un primer análisis, se utilizaron los parámetros de configuración recomendados en [Herrera et al., 1998]. Tras este análisis se establecieron qué características concretas de las funciones afectan en mayor medida al rendimiento del algoritmo. Concretamente, los tipos de funciones no recomendadas son: las funciones unimodales de camino largo, las funciones multimodales donde el centro de atracción óptimo no es el de mayor tamaño y las funciones no separables. Teniendo en cuenta la estrategia de búsqueda del algoritmo y los tipos de funciones no resueltas, se realizó un segundo análisis. Los parámetros del algoritmo y las condiciones de ejecución del procedimiento de caracterización se ajustaron para tratar de resolver estas funciones. Para ajustar estos parámetros se tuvieron en cuenta las características de balance de exploración / explotación, la presión selectiva, el paso de mutación y las direcciones de búsqueda. Se resolvieron todas las funciones analizadas excepto aquellas que son *No-Separables*. Además de la búsqueda en direcciones diagonales, este tipo de funciones requiere que los operadores de los algoritmos tengan algún tipo de mecanismo que ajuste los genes de los cromosomas teniendo en cuenta las dependencias entre los mismos. Los operadores utilizados en este RCGA no presentan esta característica, por lo tanto su rendimiento en funciones de tipo *No-Separable* será pobre.

5.2 Covariance Matrix Adaptation - Evolutionary Strategy (CMA-ES)

5.2.1 Descripción del algoritmo

El algoritmo CMA-ES es un algoritmo evolutivo propuesto originalmente para la optimización de funciones no lineales y no convexas en dominios continuos. La primera versión fue presentada por Nikolaus Hansen y Andreas Ostermeier [Hansen and Ostermeier, 2001]. Este algoritmo utiliza una distribución normal multivariada para generar la nueva población generación tras generación. El CMA-ES es una aproximación de segundo orden, es decir, mediante la adaptación de los parámetro de la matriz de covarianzas es capaz de extraer las dependencias de las variables del poblema y adaptarlas de acuerdo con ellas durante el proceso de optimización. Su comportamiento se basa en estimar, mediante un proceso iterativo, la matriz de covarianzas de una distribución normal multivariada que aproxima los parámetros del problema. Esta matriz está muy relacionada con la inversa de la matriz Hessiana. En optimización, esta matriz se utiliza para determinar los puntos críticos de una función utilizando las segundas derivadas. Sin embargo, en muchos problemas no es posible realizar su cálculo, bien porque la función no es derivable o bien porque el número de parámetros es elevado y, por lo tanto, el cálculo de dicha matriz requiere tiempos de cómputo muy elevados. La matriz de covarianzas permite ajustar la distribución utilizada durante el proceso de búsqueda al contorno de la función a optimizar. Al utilizar la matriz de covarianzas y no la matriz Hessiana, el algoritmo no utiliza o aproxima gradientes y, además, no requiere que estos existan. El CMA-ES obtiene resultados satisfactorios en problemas discontinuos, multimodales, con ruido y cuya superficie de búsqueda es rugosa. Además, es invariante a transformaciones de la superficie de calidad en cuanto a rotaciones, reflexiones o traslaciones.

Como ya se ha mencionado anteriormente, en cada iteración el algoritmo genera una nueva población a partir de la información de una distribución normal multivariada. Esta distribución se define como:

$$x_k \sim N(m, \sigma^2 C), k = 1, \ldots, n \tag{5.3}$$

donde:

- x_k, es el individuo k-ésimo de la población.
- m, es la media de la distribución de búsqueda.
- σ^2, es la desviación *total* o la longitud del paso de búsqueda.
- C, es la matriz de covarianza.
- n, es el tamaño de la población.

Después de la generación de una nueva población, los parámetros m, σ y C se actualizan utilizando la población actual. Esta actualización es clave en el funcionamiento del algoritmo y a continuación se describe de manera detallada:

Cálculo de m: selección y reproducción

Los procesos de selección y reproducción definidos para los AEs en el capítulo 3 se corresponden, en el caso del algoritmo CMA-ES, con el proceso de actualización de la media m de la distribución. La nueva media se obtiene como una media ponderada de los μ mejores individuos de la población. Es un mecanismo de selección no elitista, ya que estos μ puntos no se conservan para la generación siguiente. El cálculo de $\mathbf{m}^{(g+1)}$ se realiza según la siguiente ecuación:

$$\mathbf{m}^{(g+1)} = \sum_{i=1}^{\mu} w_i \mathbf{x}_{i:\lambda}^{(g+1)}, \sum_{i=1}^{\mu} w_i = 1, \; w_i > 0, \text{for } i = 1, \ldots, \mu \tag{5.4}$$

donde:

- $\mu \leq \lambda$ es el número de puntos seleccionados para calcular la nueva media, es decir, el conjunto de padres a partir del cual se obtiene la información para generar la población de hijos.
- $w_{i=1,\ldots,\mu} \in \mathscr{R}_+$, son los pesos utilizados para la media ponderada.
- $\mathbf{x}_{i:\lambda}^{(g+1)}$, es el mejor i-ésimo individuo de la población.

Control del paso de búsqueda

El siguiente paso del algoritmo consiste en el control de la longitud del paso de búsqueda. Este control es necesario por diversas razones. En primer lugar, porque no es posible aproximar de manera correcta la longitud óptima del paso de búsqueda para cada función a optimizar. En segundo lugar, debido a que la tasa de aprendizaje más fiable utilizada para actualizar la matriz de covarianzas provoca que los cambios en el paso de búsqueda sean demasiado lentos. De esta forma, el algoritmo pierde desde el punto de vista competitivo, es decir, cuando se compara su velocidad de convergencia con otros algoritmos. Por último, el control del paso de búsqueda es necesario ya que la actualización de la matriz de covarianzas puede provocar que la varianza se incremente simultáneamente en todas las direcciones.

Para controlar la longitud del paso, el algoritmo utiliza el camino de evolución, es decir, la suma de los pasos que han sido realizados hasta dicho instante de la evolución. Este método puede aplicarse independientemente de la actualización de la matriz de covarianza y se denomina "control acumulativo de la longitud del camino". La longitud de un camino de evolución se explota basándose en el siguiente razonamiento:

- Si el camino de evolución es largo, todos los pasos individuales están apuntando en la misma dirección de búsqueda, es decir, están correlacionados. En este caso, la longitud del paso de búsqueda debe ser incrementada, ya que el algoritmo podría cubrir la misma distancia en un menor número de pasos.
- En el caso contrario, si el camino de evolución es corto, los pasos individuales se están cancelando entre sí, es decir, están anti-correlados. Por lo tanto, la longitud del paso deberá ser decrementada.

Para definir camino largo y corto, los autores comparan la longitud del camino de evolución con la longitud esperada si se realizase una selección aleatoria.

Actualización de la matriz de covarianzas: *C*

El último paso del algoritmo es la actualización de la matriz de covarianza basándose en [Hansen and Ostermeier, 1996]. Esta matriz representa una estimación de la distribución de los pasos seleccionados. De esta forma, generando la nueva población a partir de la información de esta matriz podemos suponer que el algoritmo realizará pasos de búsqueda en direcciones satisfactorias. Para actualizar esta matriz, el algoritmo aprende las dependencias entre variables. Estas dependencias se reflejan en los elementos no diagonales de la matriz.

Este algoritmo ha sido aplicado con éxito a la resolución de multitud de problemas de optimización [Hansen and Kern, 2004, Hansen et al., 2003] y a problemas reales [Hansen et al., 2009, Abudhahir and Baskar, 2008][2].

Este algoritmo no es el único en el campo de la CE que utiliza distribuciones de probabilidad para adaptar las variables de los problemas a soluciones, los EDAs también utilizan un mecanismo de búsqueda similar a éste para la optimización de funciones. Aunque, los EDAs comparten características con el CMA-ES, y éste podría considerarse un tipo de EDA, también existen diferencias significativas en sus implemenaciones [Hansen, 2006]. Por ejemplo:

- En cuanto a la forma que ambos algoritmos estiman la distribución de probabilidad, los EDAs lo hacen a partir de ciertos individuos seleccionados. El CMA-ES estima la ditribución basándose en los pasos realizados por los individuos, de esta forma la probabilidad de convergencia prematura es menor.

- En los algoritmos de tipo EDA generalmente no se realiza un control de la longitud del paso de mutación, en el CMA-ES este control sí que se lleva a cabo. Para ello se utiliza la longitud del camino recorrido, este mecanismo mejora la velocidad de convergencia y las capacidades de búsqueda del algoritmo.

- La estimación de la distribución en el CMA-ES es independiente del tamaño de la población, en los EDAs no.

- La propiedad de invarianza ante rotaciones, escalado o traslación que presenta el algoritmo CMA-ES no se cumple en los EDAs debido a que estos algoritmos utilizan el sistema de coordenadas para estimar la distribución de probabilidad.

También existen EEs que utilizan distribuciones normales para ajustar los parámetros del problema, sin embargo, este tipo de estrategias utilizan procesos estocásticos para modificar las distribuciones de estos parámetros, mientras que en el CMA-ES estos cambios son determinísticos y están relacionadas con las variaciones de los propios parámetros del problema.

Para resumir el comportamiento del CMA-ES, estas son sus características principales:

- Se utiliza una distribución normal multivariada para generar a nueva población utilizando el principio de máxima entropía.

- En este algoritmo el proceso de selección, entendido como la elección de los individuos que se utilizarán para estimar la matriz de covarianzas, se basa en la clasificación de los individuos en términos de calidad y se utiliza una media ponderada para la reproducción.

[2]En *www.lri.fr/ hansen/cmaapplications.pdf* se encuentra una lista actualizada de aplicaciones reales del CMA-ES.

- La adaptación de la matriz de covarianzas incrementa la probabilidad de repetir pasos exitosos.
- El control de la longitud del camino de evolución controla la longitud del paso de búsqueda utilizando el camino de evolución.

En este trabajo se utilizará una versión del CMA-ES que incluye mecanismos de reinicio y de incremento de la población [Auger and Hansen, 2005]. En esta versión del algoritmo se establecen una serie de criterios y, cuando alguno de estos criterios se cumple, se realiza un reinicio del algoritmo incrementando el tamaño de la población. Se han aplicado los mismos criterios de reinicio que en el artículo original, estos son:

- Si el cambio total en la función objetivo durante $10 + \lfloor 30n/\lambda \rfloor$ generaciones es menor que γ. En las ejecuciones de este trabajo el valor de γ es $1.0e{-}12$.
- Si la desviación estándar de la distribución normal es menor que τ en todas las coordenadas y si $\sigma\vec{p_c}$ (el camino de evolución) es menor que τ en todos sus componentes. Para este trabajo el valor de τ es de $1.0e{-}6$.
- Si el número de condición de la matriz de covarianza supera el valor de 10^{14}.

El cumplimiento de una de estas condiciones es suficiente para que se produzca el reinicio del algoritmo. En resumen, el algoritmo 4 muestra el pseudocódigo de la versión concreta del CMA-ES que será analizada en el siguiente apartado. La implementación del mismo está disponible en la librería JEAF (www.gii.udc.es/jeaf).

5.2.2 Caracterización del algoritmo CMA-ES

En las dos primeras etapas del procedimiento de caracterización se ha seleccionado el conjunto de funciones prueba caracterizado en la sección 4.1.2 cuyas propiedades se resumen en las tablas 4.3, 4.4 y 4.5 y, en cuanto a las medidas de error y rendimiento, se utilizarán el CPEM, el SR y el G_{NE}. Describimos a continuación la tercera etapa del procedimiento propuesto, centrada en el estudio experimental y compuesta por la ejecución del algoritmo y el análisis de resultados.

Ejecución de las pruebas

El primer paso para la caracterización del algoritmo consiste en ejecutar el algoritmo sobre el conjunto de prueba siguiendo las indicaciones explicadas en 4.3.1. Los parámetros utilizados para ejecutar el CMA-ES se muestran en la tabla 5.6 y han sido seleccionados siguiendo las indicaciones de los autores en [Auger and Hansen, 2005]. Como parte de este tercer paso se llevó a cabo un análisis poblacional con el objetivo de determinar el tamaño de población óptimo para cada función. En el siguiente apartado se analizan los resultados obtenidos en este sentido.

Análisis poblacional El tamaño de población óptimo para cada función del conjunto de prueba se muestra en la tercera columna de la tabla 5.8. Analizando los resultados obtenidos podemos observar que para las funciones unimodales el número de individuos requeridos es, en general, menor que para las funciones multimodales. Así, la mayoría de las funciones

Algoritmo 4 Esquema básico la estrategia CMA-ES.

Entrada:

NP, número de individuos de la población.
D, número de genes de cada cromosoma.
t_{max}, número máximo de generaciones.
$m, \sigma \in \Re$, se generan aleatoriamente

$t \leftarrow 0$
$C \leftarrow I$
$p_c \leftarrow 0, p_\sigma \leftarrow 0$
$c_c, c_\sigma \approx \frac{4}{NP}, c_{cov} \approx \frac{2}{NP^2}$
$\mu_{eff} \approx 0.3\lambda, \mu_{cov} = \mu_{eff}$
$d_\sigma \approx 1 + \sqrt{\frac{\mu_{eff}}{NP}}$
Generar la población inicial de forma aleatoria: $\mathbf{x}_t^i \sim N_i(m_t, \sigma_t^2 C_t), \forall i, i = 1, \ldots, NP$
Evaluar $f(\mathbf{x}_t^i), \forall i, i = 1, \ldots, NP$
for $t = 1$ hasta t_{max} **do**
  $t \leftarrow t + 1$
  // Actualización de los parámetros de la distribución normal
  $\langle z \rangle_{sel} = \sum_{i=1}^{\mu} w_i x_i, w_1, \ldots, w_\mu \geq 0, \sum_{i=1}^{\mu} w_i = 1$
  $m_t \leftarrow m_{t-1} + \sigma \langle z \rangle_{sel}$
  $p_c \leftarrow (1 - c_c) p_c + \sqrt{1 - (1_c c)^2} \sqrt{\mu_{eff}} \langle z \rangle_{sel}$
  $C_t \leftarrow (1 - c_{cov}) C_{t-1} + c_{cov} p_c p_c^T$
  $p_\sigma \leftarrow (1 - c_\sigma) p_\sigma + \sqrt{1 - (1_c \sigma)^2} \sqrt{\mu_{eff}} C^{-1/2} \langle z \rangle_{sel}$
  $\sigma_t \leftarrow \sigma_{t-1} \times exp\left(\frac{c_\sigma}{d_\sigma} \left(\frac{\|p_\sigma\|}{E\|N(0,I)\|} - 1 \right) \right)$
  // Se genera la población de la siguiente generación
  $\mathbf{x}_t^i \sim N_i(m_t, \sigma_t^2 C_t), \forall i, i = 1, \ldots, NP$
end for

unimodales requieren un tamaño de población de 10 o 20 individuos, excepto la función *Schwefel 2.21* para dimesión 10 y 30, la función *Kowaliks* de dimensión 4 y la función *Perm* para dimensión 30 y 50. En la figura 5.8, se muestra el valor medio de CPEM obtenido para estas funciones en los tamaños de población analizados (de 10 a 100 individuos). Como se puede observar en las gráficas, el valor de CPEM en estas excepciones es indepediente del número de individuos utilizado.

En el caso de las funciones multimodales, en un primer análisis no se puede extraer una tendencia general. Todas las funciones de baja dimensionalidad mantienen la tendencia de las funciones unimodales requiriendo únicamente 10 o 20 individuos para obtener los mejores resultados en cada función con tres excepciones: las funciones *Shekel 5*, *Shekel 7* y *Shekel*

Parámetro	Valor
X inicial	0.0
Desviación estándar inicial	1.0
Factor incr. población	2.0

Tabla 5.6: Parámetros de configuración utilizados para la caracterización del CMA-ES.

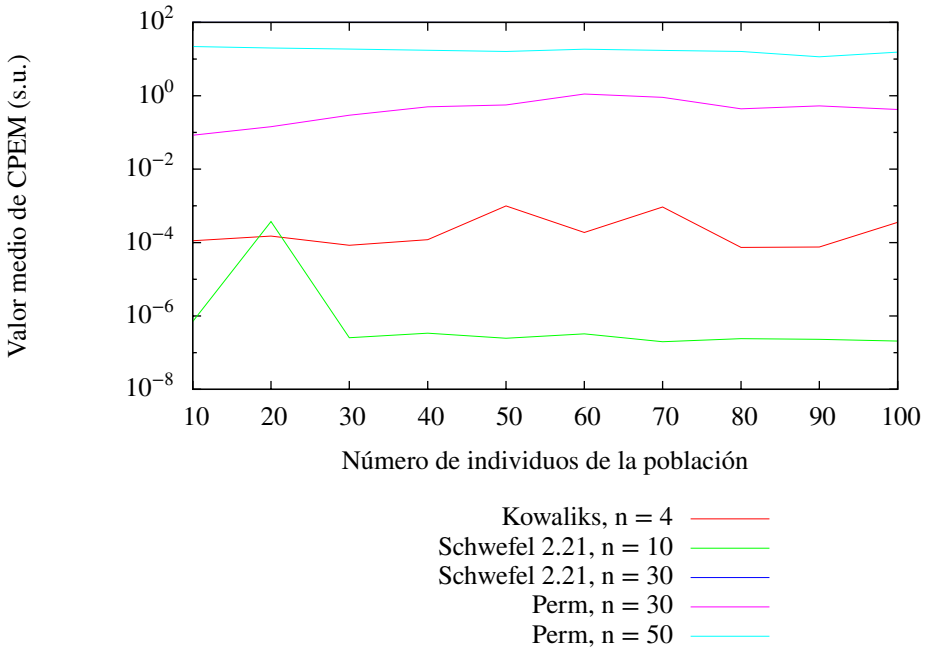


Figura 5.8: Análisis poblacional del algoritmo CMA-ES para funciones unimodales

10. Como se explicará más adelante, la topología de estas funciones provoca que el CMA-ES tienda a converger hacia óptimos locales. Teniendo en cuenta los resultados de la tabla 5.7, donde se presentan los datos del análisis poblacional de la familia de funciones *Shekel*, analizando cada una de las tres funciones se obtiene la conclusión de que cualquier tamaño de población proporciona los mismos resultados. En la función *Shekel 5*, el nivel de error en todas las poblaciones es similar. En el caso de las funciones *Shekel 7* y *Shekel 10*, hay que tener en cuenta los valores de *CPEM* y SR conjuntamente para realizar el análisis del tamaño de la población. A la vista de estos resultados, todas los tamaños de población obtienen valores similares. Por lo tanto, en este tipo de funciones, a igualdad de evaluaciones, el CMA-ES se beneficia más de un mayor número de generaciones, para lo cual es necesario un tamaño de población bajo, que de un tamaño de población mayor.

Las funciones multimodales de alta dimensionalidad, como son las funciones *Cosine*, *Rastrigin*, *Schwefel*, *Griewank*, *Levy*, *Penalized 1*, *Penalized 2* y *Rosenbrock*, requieren un mayor nivel de exploración. Por este motivo, el número de individuos requeridos para resolver este tipo de funciones es mayor que en las funciones unimodales. En este grupo, las funciones *Cosine Mixture* y *Levy* de dimensiones 30 y 50, *Griewank* de dimensión 10, *Rastrigin* y *Schwefel*, nunca se resuelven, por lo tanto el tamaño de población requerido no es un dato a tener en cuenta. En los casos en los que sí se resuelven con más de un 75% de SR, como son la función *Cosine* y *Levy* de dimensión 10, *Griewank* de dimensión 30 y 50, *Penalized 1*, *Penalized 2* y *Rosenbrock*; el número de individuos requerido es mayor que en el resto de funciones debido a la multimodalidad de las funciones.

Resumiendo, en el caso del CMA-ES implementado, cuando la función a resolver es unimodal o multimodal de baja dimensionalidad el número de individuos requerido es inferior a 20 individuos. En caso contrario, cuando las funciones que se trata de resolver son multimodales y de alta dimensionalidad, al necesitar un mayor nivel de exploración, el número de individuos que utiliza es mayor.

Población	Shekel 5		Shekel 7		Shekel 10	
	CPEM	SR	CPEM	SR	CPEM	SR
10	$1.19e{+}0$	0.72	$2.13e{-}1$	0.96	$2.14e{-}1$	0.96
20	$2.32e{-}1$	0.60	$2.31e{-}1$	0.88	$2.35e{-}1$	0.84
30	$1.42e{+}0$	0.72	$4.11e{-}1$	0.92	$1.34e{-}7$	1.00
40	$1.41e{+}0$	0.76	$2.67e{-}1$	0.96	$1.89e{-}7$	1.00
50	$1.82e{+}0$	0.64	$1.33e{-}7$	1.00	$3.07e{-}1$	0.96
60	$1.42e{+}0$	0.72	$4.50e{-}6$	0.96	$3.09e{-}1$	0.96
70	$7.03e{-}1$	0.88	$2.13e{-}1$	0.96	$1.79e{-}7$	1.00
80	$7.05e{-}1$	0.52	$2.13e{-}1$	0.96	$2.68e{-}1$	0.92
90	$1.19e{+}0$	0.56	$8.99e{-}3$	0.88	$2.15e{-}1$	0.84
100	$4.20e{-}1$	0.72	$7.32e{-}1$	0.76	$3.80e{-}3$	0.92

Tabla 5.7: Análisis poblacional de las funciones pertenecientes a la familia *Shekel*

Análisis de resultados

Al igual que en la sección anterior, después de ejecutar el CMA-ES sobre el conjunto de funciones prueba utilizando las condiciones detalladas la sección anterior, se obtuvieron los resultados que se presentan en la tabla 5.8. De nuevo, las columnas de la tabla muestran, de izquierda a derecha, el nombre de cada función, su dimensión, el tamaño de la población que obtuvo los mejores resultados, la tasa de éxito, el valor de *CPEM* obtenido y el error genotípico.

Tabla 5.8: Resultados obtenidos por el algoritmo CMA-ES en el conjunto de funciones utilizadas en este trabajo.

Function	D	N	SR	CPEM	GNE
Axis Parallel	10	10	1.00	$2.11e{-}8 \pm 8.54e{-}10$	$2.28e{-}6 \pm 4.17e{-}7$
	30	10	1.00	$2.58e{-}8 \pm 8.37e{-}10$	$8.02e{-}7 \pm 1.19e{-}7$
	50	10	1.00	$3.03e{-}8 \pm 1.09e{-}9$	$5.83e{-}7 \pm 1.58e{-}7$
Schwefel 2.22	10	10	1.00	$3.27e{-}8 \pm 1.90e{-}9$	$1.64e{-}8 \pm 2.67e{-}9$
	30	20	1.00	$5.83e{-}8 \pm 7.36e{-}9$	$6.31e{-}9 \pm 3.05e{-}9$
	50	40	1.00	$1.09e{-}7 \pm 2.88e{-}8$	$3.64e{-}9 \pm 1.89e{-}9$
Sphere	10	10	1.00	$1.82e{-}8 \pm 7.44e{-}10$	$4.23e{-}6 \pm 9.76e{-}7$
	30	10	1.00	$1.76e{-}8 \pm 5.05e{-}10$	$1.96e{-}6 \pm 9.11e{-}8$
	50	10	1.00	$1.74e{-}8 \pm 3.09e{-}10$	$1.49e{-}6 \pm 2.66e{-}8$
Step	10	10	1.00	$9.92e{-}9 \pm 1.77e{-}9$	$3.86e{-}3 \pm 7.02e{-}4$
	30	10	1.00	$2.44e{-}8 \pm 1.03e{-}8$	$2.89e{-}3 \pm 1.83e{-}4$
	50	20	1.00	$4.99e{-}8 \pm 2.49e{-}8$	$2.80e{-}2 \pm 1.24e{-}1$
SumOf	10	10	1.00	$1.24e{-}7 \pm 1.77e{-}8$	$8.96e{-}2 \pm 1.38e{-}2$
	30	10	1.00	$2.89e{-}7 \pm 2.76e{-}8$	$2.70e{-}1 \pm 2.06e{-}2$
	50	10	1.00	$4.46e{-}7 \pm 2.76e{-}8$	$3.45e{-}1 \pm 2.05e{-}2$
Easom	2	10	0.80	$1.95e{-}1 \pm 3.91e{-}1$	$1.21e{-}1 \pm 3.31e{-}1$
Schwefel 2.21	10	70	1.00	$1.99e{-}7 \pm 5.80e{-}8$	$8.61e{-}9 \pm 3.28e{-}9$
	30	90	0.00	$9.88e{+}1 \pm 3.96e{+}0$	$9.60e{-}1 \pm 2.64e{-}2$
	50	10	0.00	$1.00e{+}2 \pm 0.00e{+}0$	$9.74e{-}1 \pm 1.50e{-}2$
Colville	4	10	1.00	$4.81e{-}8 \pm 1.16e{-}8$	$7.09e{-}5 \pm 3.40e{-}5$
Matyas	2	10	1.00	$1.40e{-}8 \pm 1.67e{-}9$	$8.22e{-}4 \pm 3.80e{-}4$
Perm	10	10	0.32	$8.40e{-}2 \pm 2.16e{-}1$	$4.29e{-}2 \pm 7.20e{-}2$
	30	50	0.12	$1.61e{+}1 \pm 1.95e{+}1$	$6.54e{-}3 \pm 1.97e{-}3$
	50	60	0.08	$2.26e{+}1 \pm 2.99e{+}1$	$3.20e{-}3 \pm 9.60e{-}4$

Tabla 5.8 -- continua desde la página anterior

Function	D	N	SR	CPEM	GNE
Schwefel 1.2	10	10	1.00	$2.74e{-8} \pm 1.36e{-9}$	$6.09e{-6} \pm 1.82e{-6}$
	30	10	1.00	$5.08e{-8} \pm 1.55e{-9}$	$2.87e{-6} \pm 2.00e{-7}$
	50	10	1.00	$7.19e{-8} \pm 1.81e{-9}$	$2.34e{-6} \pm 9.37e{-8}$
Zakharov	10	10	1.00	$2.46e{-8} \pm 1.78e{-9}$	$5.10e{-5} \pm 9.68e{-6}$
	30	10	1.00	$5.52e{-8} \pm 2.20e{-9}$	$2.54e{-5} \pm 9.07e{-7}$
	50	10	1.00	$8.99e{-8} \pm 2.99e{-9}$	$1.92e{-5} \pm 1.87e{-7}$
Aluffi-Pentinis	2	20	1.00	$2.57e{-8} \pm 1.97e{-9}$	$2.20e{-4} \pm 1.24e{-4}$
Becker and Lago	2	10	1.00	$1.61e{-8} \pm 1.68e{-9}$	$5.91e{-1} \pm 3.96e{-1}$
Bohachevsky 1	2	20	1.00	$3.56e{-8} \pm 3.33e{-9}$	$1.61e{-5} \pm 4.06e{-5}$
Cosine Mixture	10	80	0.92	$1.18e{-2} \pm 4.01e{-2}$	$9.43e{-3} \pm 3.23e{-2}$
	30	80	0.04	$5.20e{-1} \pm 2.78e{-1}$	$1.31e{-1} \pm 4.16e{-2}$
	50	100	0.00	$1.41e{+0} \pm 5.45e{-1}$	$1.82e{-1} \pm 9.01e{-2}$
Rastrigin	10	80	0.00	$1.09e{+1} \pm 3.58e{+0}$	$2.13e{-1} \pm 5.63e{-2}$
	30	80	0.00	$5.86e{+1} \pm 1.45e{+1}$	$2.80e{-1} \pm 4.07e{-2}$
	50	100	0.00	$1.50e{+2} \pm 1.72e{+1}$	$3.52e{-1} \pm 3.41e{-2}$
Schwefel	10	10	0.00	$8.98e{+2} \pm 2.26e{+2}$	$1.03e{+0} \pm 1.85e{-1}$
	30	60	0.00	$3.69e{+3} \pm 5.91e{+2}$	$1.01e{+0} \pm 1.17e{-1}$
	50	100	0.00	$6.63e{+3} \pm 7.60e{+2}$	$1.01e{+0} \pm 7.26e{-2}$
Ackleys	10	20	1.00	$5.31e{-8} \pm 3.67e{-8}$	$1.05e{-8} \pm 1.74e{-9}$
	30	40	1.00	$1.01e{-7} \pm 6.26e{-8}$	$8.78e{-9} \pm 6.06e{-10}$
	50	50	1.00	$1.49e{-7} \pm 8.97e{-8}$	$8.35e{-9} \pm 2.84e{-10}$
Griewank	30	50	0.88	$9.86e{-4} \pm 2.70e{-3}$	$5.85e{-2} \pm 1.62e{-1}$
	50	50	1.00	$1.91e{-7} \pm 2.12e{-7}$	$1.74e{-6} \pm 6.70e{-8}$
Levy	10	40	0.96	$7.92e{-2} \pm 3.88e{-1}$	$7.24e{-3} \pm 3.54e{-2}$
	30	70	0.04	$3.33e{+0} \pm 1.87e{+0}$	$1.40e{-1} \pm 6.91e{-2}$
	50	50	0.00	$1.50e{+1} \pm 6.90e{+0}$	$2.22e{-1} \pm 5.24e{-2}$
Penalized 1	10	40	1.00	$6.81e{-8} \pm 5.19e{-8}$	$5.61e{-5} \pm 9.92e{-6}$
	30	50	1.00	$1.40e{-7} \pm 1.65e{-7}$	$4.86e{-5} \pm 2.63e{-6}$
	50	100	0.92	$4.52e{-3} \pm 1.54e{-2}$	$7.67e{-4} \pm 2.50e{-3}$
Penalized 2	10	30	1.00	$7.81e{-8} \pm 1.79e{-7}$	$2.38e{-5} \pm 4.64e{-6}$
	30	90	0.80	$2.20e{-3} \pm 4.39e{-3}$	$2.51e{-4} \pm 4.86e{-4}$
	50	90	0.76	$2.64e{-3} \pm 4.69e{-3}$	$2.35e{-4} \pm 4.00e{-4}$
Beale	2	10	1.00	$3.77e{-8} \pm 3.74e{-8}$	$1.34e{-1} \pm 3.69e{-1}$
Bohachevsky 2	2	20	1.00	$3.52e{-8} \pm 1.93e{-9}$	$8.66e{-6} \pm 5.11e{-6}$
Dekkers and Aarts	2	10	1.00	$1.41e{-8} \pm 2.95e{-9}$	$2.92e{-1} \pm 4.69e{-1}$
Goldstein Price	2	10	1.00	$3.01e{-8} \pm 5.97e{-8}$	$7.50e{-5} \pm 1.02e{-4}$
Griewank	10	10	0.04	$3.20e{-2} \pm 1.89e{-2}$	$1.24e{-2} \pm 3.03e{-2}$
Hartman 3	3	10	1.00	$1.71e{-8} \pm 1.93e{-8}$	$8.82e{-3} \pm 5.99e{-3}$
Hartman 6	6	10	0.96	$5.06e{-3} \pm 2.48e{-2}$	$4.50e{-2} \pm 1.78e{-1}$
Rosenbrock	10	40	1.00	$2.45e{-7} \pm 1.58e{-7}$	$7.47e{-6} \pm 5.00e{-6}$
	30	100	1.00	$7.60e{-7} \pm 1.07e{-7}$	$2.82e{-6} \pm 1.85e{-6}$
	50	40	0.88	$4.78e{-1} \pm 1.30e{+0}$	$3.45e{-3} \pm 8.12e{-3}$
Kowaliks	4	80	0.80	$1.32e{-4} \pm 2.64e{-4}$	$4.89e{+0} \pm 1.21e{+1}$
Shekel Family 5	4	70	0.88	$7.03e{-1} \pm 1.94e{+0}$	$8.00e{-2} \pm 2.31e{-1}$
Shekel Family 7	4	50	1.00	$1.33e{-7} \pm 1.61e{-7}$	$2.08e{-4} \pm 6.01e{-5}$
Shekel Family 10	4	30	1.00	$1.34e{-7} \pm 1.44e{-7}$	$2.71e{-4} \pm 1.22e{-4}$
Shekels Foxholes	2	10	1.00	$5.31e{-7} \pm 2.71e{-7}$	$1.71e{-2} \pm 4.67e{-2}$
SixHump Camel Back	2	10	1.00	$1.59e{-8} \pm 1.40e{-9}$	$8.14e{-2} \pm 1.01e{-1}$

Configuración básica Como se hizo en la sección anterior dedicada al RCGA, en primer lugar se analizarán los resultados del algoritmo CMA-ES utilizando la configuración básica propuesta en [Auger and Hansen, 2005]. Observando los resultados de la tabla 5.8 de forma

general, se concluye que el CMA-ES presenta problemas con funciones multimodales de alta dimensionalidad de tipo *L-Separables*. El algoritmo falla en las tres funciones que presentan estas propiedades de forma conjunta (*Cosine*, *Rastrigin* y *Schwefel*). Aunque su rendimiento mejora en comparación con los casos anteriores, el CMA-ES también presenta problemas en funciones de tipo *NL-Separables* de alta dimensionalidad, con independencia del tipo de modalidad. Concretamente, las funciones en las cuales su rendimiento es realmente pobre son la *Schwefel 2.21* de tipo unimodal y las *Levy*, *Penalized 2* y *Griewank* de tipo multimodal. La no separabilidad de las funciones no parece ser un problema para este algoritmo, ya que, como se observa en la tabla de resultados, solamente falla en dos de estas funciones: la función *Perm* y la función *Griewank*. En el resto de funciones *No-Separables* el SR es siempre mayor del 88%. A continuación se profundizará en el comportamiento del CMA-ES centrándonos en la modalidad de las funciones.

Tras un primer análisis es posible concluir que, de forma general, el algoritmo CMA-ES presenta problemas en las funciones multimodales de alta dimensionalidad de tipo *L-Separable* y *NL-Separable*, resolviendo sin problemas casi todos los ejemplos restantes. Existen nueve funciones multimodales de alta dimensionalidad en el conjunto de funciones de prueba y el CMA-ES resuelve todas las ejecuciones de la función *Ackleys*, falla en algunas ejecuciones de tres (las funciones *Penalized 1*, *Penalized 2* y *Rosenbrock*) y falla siempre en cinco de ellas (las funciones *Rastrigin*, *Schwefel*, *Cosine Mixture*, *Levy* y *Griewank*). De las veintisiete funciones restantes, sólo falla en 2 (las funciones *Schwefel 2.21* y *Perm*) y no resuelve algunas ejecuciones de cuatro (las funciones *Kowaliks*, *Easom*, *Hartman 6* y *Shekel Familiy 5*) debido a las características topológicas del espacio de búsqueda que serán explicadas a continuación.

En primer lugar, se explicará el comportamiento del algoritmo CMA-ES en las nueve funciones multimodales de alta dimensionalidad. La función *Griewank* es una función con muchos centros de atracción, donde el más amplio no es el óptimo (ver tabla 4.4). De hecho, la diferencia entre el tamaño del centro de atracción más amplio y el óptimo es de un orden de magnitud. El problema es peor en dimensión 10 (SR 4% frente al 88% con 30 dimensiones y 100% con 50 dimensiones) porque, a medida que crece el número de dimensiones, la no separabilidad desaparece y la distancia entre centros de atracción decrece. Estas son las razones por las cuales el CMA-ES presentan problemas con dimensiones bajas en esta función. Otra función con el mismo problema es la función *Schwefel* (SR 0%), donde la diferencia entre el tamaño del centro de atracción más amplio y el óptimo es de cuatro ordenes de magnitud y además se encuentran muy alejados uno de otro.

Las funciones *Rastrigin* (0%), *Levy* (SR 96%, 4%, 0%), *Cosine* (SR 92%, 4%, 0%), *Penalized 1* (SR 100%, 100%, 92%) y *Penalized 2* (SR 100%, 80%, 76%) son funciones *L-Separable* y *NL-Separables*, hecho que no ayuda al algoritmo. El procedimiento que sigue el CMA-ES en cada generación consiste en generar una nueva población a partir de una distribución de probabilidad generada a partir de la información de covarianzas de cada parámetro, es decir, a partir de la información de las variables que existen entre ellos. El inconveniente con las funciones separables es el coste computacional, es decir, el número de llamadas a la función de calidad que necesitan para alcanzar el objetivo cuando no existen dependencias que aprender (en la matriz de covarianzas existen $(n^2 + n)/2$ que tienen que ser ajustados). Para resolver este problema, los autores del algoritmo CMA-ES publicaron recientemente una nueva variante llamada sep-CMA-ES [Ros and Hansen, 2008] que mejora los resultados en funciones separables en cuanto a velocidad de convergencia, transformando la matriz de covarianzas en una matriz diagonal en la cual solamente se ajusta la información de cada gen y no se aprenden dependencias entre ellos. Con esta nueva aproximación, los autores reducen los grados

de libertad en la matriz de covarianzas haciendo que el proceso de búsqueda sea menos complejo. Obviamente, como se explica en [Ros and Hansen, 2008], esta variante empeora el rendimiento del algoritmo en las funciones no separables. El bajo rendimiento del CMA-ES en funciones *L-Separables* o *NL-Separables* solamente se da en funciones de alta dimensionalidad. Resulta obvio que debido a que el tamaño de la matriz de covarianzas depende de las dimensiones del problema, en estos casos el coste computacional de adaptar esta matriz sea menor.

Para terminar el análisis en funciones multimodales de alta dimensionalidad, el algoritmo también tiene problemas para resolver la función *Rosenbrock* de dimensión 50 (SR 88%) que es *No-Separable*. La función *Rosenbrock* tiene nueve centros de atracción, dos de ellos ocupan casi la totalidad del espacio de calidad y cuando la dimensionalidad crece, aumenta también la dificultad del problema para el CMA-ES. La única función multimodal de alta dimensionalidad donde el CMA-ES no tiene problemas es la función *Ackleys*. Esta función tiene un gran número de centros de atracción. De la misma manera que ocurre con el AG, la distribución de los centros de atracción facilita la búsqueda del óptimo. Este hecho refuerza la hipótesis de que no es solamente el número de centros de atracción la característica que hay que tener en cuenta a la hora de estimar la dificultad de los espacios de calidad.

Resumiendo, el rendimiento del CMA-ES ante problemas multimodales de alta dimensionalidad no es bueno cuando los centros de atracción de los óptimos locales están espaciados o cuando el centro de atracción más amplio no es el óptimo. Este mal comportamiento se debe al hecho de utilizar una distribución normal multivariada para modelar el espacio de calidad.

Teniendo en cuenta el resto de funciones, es decir unimodales o multimodales de baja dimensionalidad, el CMA-ES sólo falla claramente en 2 de ellas (las funciones *Perm* y *Schwefel 2.21*) no resolviendo ninguna ejecución, resuelve algunas ejecuciones de 4 (las funciones *Kowaliks*, *Easom*, *Hartman 6* y *Shekel Familiy 5*) y no presenta ningún problema con las otras 21 funciones. Falla en la función *Perm*, una función unimodal pero de gran dificultad debido a la longitud del camino hasta el óptimo (tabla 4.5). También falla en la función *Schwefel 2.21* (excepto en dimensión 10) por la misma razón que la función *Perm*. Las funciones *Hartman 6* (SR 96%), *Shekel Family 5* (SR 88%) y *Kowaliks* (SR 80%) tienen el mismo problema que la funciones *Griewank* y *Schwefel* (el centro de atracción más amplio no es el centro de atracción óptimo), pero no son de alta dimensionalidad y no hay mucha diferencia en los tamaños de los centros de atracción, por lo tanto, el rendimiento del CMA-ES se ve menos afectado en estas dos funciones. La función *Easom* (SR 80%) es una función de dos dimensiones con un espacio de calidad muy peculiar: presenta una amplia superficie plana (casi el 90% del espacio total) con un pico en el medio y, por esta razón, el comportamiento del CMA-ES no es totalmente satisfactorio y falla en algunas de las ejecuciones. Esto se debe a que en el área plana el algoritmo no tiene información para crear de manera adecuada la matriz de covarianzas.

Para terminar con la caracterización básica de este algoritmo, del análisis de los resultados se pueden extraer las siguientes conclusiones:

- En general, su rendimiento no es muy bueno en funciones de tipo *L-Separables* y *NL-Separables* de alta dimensionalidad.

- Su rendimiento es muy bueno en funciones no separables. Para resolver este tipo de funciones, un algoritmo debe aprender las dependencias que existen entre los parámetros. El éxito del CMA-ES en este tipo de funciones es una consecuencia del uso de la matriz de covarianzas para generar la nueva población. Esta matriz implícitamente

aprende las dependencias entre parámetros y explota esta información.

- En las funciones unimodales, un camino hasta el óptimo largo puede hacer que el CMA-ES falle. Esto solamente ocurre en la función *Perm* (no separable) y *Schwefel 2.21* (*NL-Separable*) y es una consecuencia del ajuste del paso de mutación que realiza el CMA-ES durante el proceso de búsqueda, como se describe en [Hansen, 2007].

- El principal problema del CMA-ES aparece cuando se unen las propiedades de multimodalidad y alta dimensionalidad. El algoritmo tiene problemas en casi todas las funciones del conjunto de prueba que presentan estas dos características (hay solamente una excepción). Si una función tiene varios centros de atracción distribuidos por el espacio de calidad y el número de dimensiones es mayor que 10, el CMA-ES probablemente no obtenga buenos resultados. Cuanto mayor es la distancia entre los centros de atracción, más difícil es el problema. Si hay un centro de atracción cuyo tamaño es mayor que el del centro de atracción óptimo, el problema es todavía más complejo para el algoritmo. Este último problema aparece incluso en funciones de baja dimensionalidad.

Variaciones sobre la configuración básica De la misma forma que en la sección anterior donde, tras el análisis de los resultados obtenidos por el RCGA con la configuración recomendada por los autores, se realizó un post-análisis tratando de mejorar el rendimiento, en esta sección se llevará a cabo el mismo post-análisis con el algoritmo CMA-ES. Para ello, en primer lugar, se identificaran las funciones donde el comportamiento del algoritmo es destacable por sus malos resultados, con SR menores al 50%. En este caso las funciones a analizar son las siguientes, divididas según su modalidad:

- Las funciones unimodales *Perm* y *Schwefel 2.21*, ambas con un camino al óptimo muy largo. La primera de ellas *No-Separable* y la segunda *NL-Separable*.

- Las funciones multimodales *Cosine*, *Rastrigin*, *Schwefel* y *Levy*, todas ellas de alta dimensionalidad. Las tres primeras son *L-Separables* y la última *NL-Separable*.

El problema del CMA-ES con las funciones separables es conocido, como ya hemos mencionado en el análisis de los resultados presentado en el apartado anterior, existe una versión de este algoritmo para funciones separables [Ros and Hansen, 2008]. Esta versión consiste en utilizar una matriz diagonal, esto implica un menor coste computacional y acelera la velocidad de convergencia del algoritmo en funciones de tipo *L-Separables*. Al ser una implementación diferente del algoritmo presentado en este trabajo y no una simple variación de un parámetro, no entra dentro del alcance de este apartado implementarlo y analizar el comportamiento del nuevo algoritmo. Por tanto, podemos concluir que esta versión del CMA-ES no es la óptima para trabajar con espacios de calidad de tipo *L-Separables* como los de las funciones *Rastrigin*, *Schwefel* o *Cosine*.

Las tres funciones restantes son de tipo *NL-Separable* (las funciones *Levy* y *Schwefel 2.21*) y *No-Separable* (la función *Perm*). Esta última función es unimodal, y como ya se ha mencionado anteriormente, es de camino muy largo. Este tipo de funciones suele requerir un nivel de explotación alto. El CMA-ES tiene un nivel de explotación alto, sin embargo, en este caso, con dicho nivel de explotación la población converge rápidamente y pierde variabilidad para generar la matriz de covarianzas. Para incrementar el nivel de exploración y aumentar dicha variabilidad se debe incrementar el valor de μ. Los resultados obtenidos al ejecutar el

D	N	μ	SR	CPEM	G_{NE}
10	10	1	0.32	$8.40e{-2} \pm 2.16e{-1}$	$4.29e{-2} \pm 7.20e{-2}$
		2	0.32	$9.07e{-5} \pm 1.72e{-4}$	$1.62e{-2} \pm 9.32e{-3}$
30	50	1	0.12	$1.61e{+1} \pm 1.95e{+1}$	$6.54e{-3} \pm 1.97e{-3}$
		2	0.00	$2.12e{+0} \pm 6.22e{+0}$	$7.19e{-2} \pm 1.45e{-1}$
50	60	1	0.08	$2.26e{+1} \pm 2.99e{+1}$	$3.20e{-3} \pm 9.60e{-4}$
		2	0.04	$1.20e{+1} \pm 1.66e{+1}$	$3.23e{-2} \pm 1.28e{-1}$

Tabla 5.9: Comparación de los resultados del CMA-ES para la función *Perm* incrementando el valor de μ de 1 a 20 individuos.

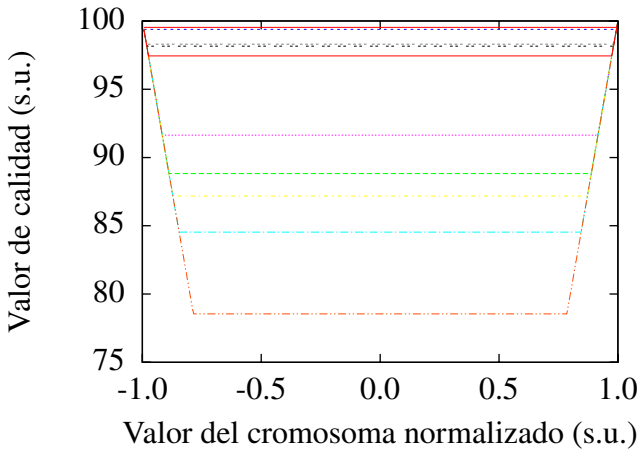


Figura 5.9: Análisis de separabilidad: Función *Schwefel 2.21*.

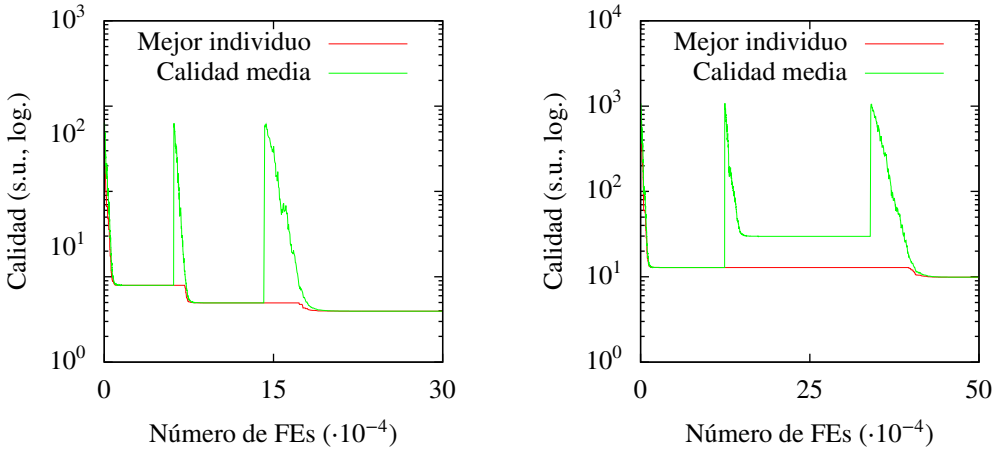
CMA-ES sobre la función *Perm* con esta nueva configuración se muestran en la tabla 5.9, donde se observa como los resultados obtenidos mejoran en términos de SR, sobre todo, en dimensión 10 y 30.

El problema de la función *Schwefel 2.21* es parecido al anterior, ya que es una función unimodal y de camino largo. En la figura 5.9 se muestra el análisis de separabilidad de esta función. Como muestra dicha figura dependiendo del valor de los genes se generan superficies planas. Dichas superficies planas carecen de información de gradiente que permitan generar de forma adecuada la matriz de covarianzas. De nuevo, aumentar el valor de μ haría que aumente el nivel de exploración mejorando la configuración original del algoritmo. Los resultados obtenidos tras ejecutar el CMA-ES con la nueva configuración se muestran en la tabla 5.10. Como se observa, el valor de μ es elevado, concretamente de 20 individuos para dimensión 30 y 45 para dimensión 50. Utilizando este valor el nivel de exploración de esta nueva configuración, se mejoran los resultados obtenidos, tanto en términos de SR como de CPEM en dimensión 30. No ocurre lo mismo para dimensión 50, donde la mejora de los resultados es mínima.

Para finalizar, analizando los resultados de las ejecuciones originales de la función *Levy*, es posible concluir que el nivel de explotación es elevado para esta función. Esta conclusión se obtiene a partir de las gráficas de evolución, donde la población converge muy rápidamente perdiendo variabilidad, como se muestra en la figura 5.10 donde se muestran dos gráficas de evolución de la función Levy, con dimensión 30 (figura 5.10a) y con dimensión 50 (figura 5.10b). Para solucionar este problema es necesario incrementar el nivel de exploración aumentando el valor de μ como en las funciones anteriores. Los nuevos resultados, comparados con los obtenidos con la configuración original, se muestran en la tabla 5.11. Incrementando

D	N	μ	SR	CPEM	G_{NE}
30	90	1	0.00	$9.88e{+}1 \pm 3.96e{+}0$	$9.60e{-}1 \pm 2.64e{-}2$
		20	0.56	$2.49e{-}1 \pm 2.83e{-}1$	$1.37e{-}3 \pm 1.56e{-}3$
50	10	1	0.00	$1.00e{+}2 \pm 0.00e{+}0$	$9.74e{-}1 \pm 1.50e{-}2$
		45	0.00	$8.80e{+}1 \pm 3.24e{+}1$	$8.53e{-}1 \pm 3.15e{-}1$

Tabla 5.10: Comparación de los resultados del CMA-ES para la función *Schwefel 2.21* incrementando el valor de μ de 1 a 20 individuos en dimensión 30 y de 1 a 45 individuos en dimensión 50.



(a) Dimensión 30 (b) Dimensión 50

Figura 5.10: Gráficas de evolución de la función *Levy* para dimensión 30 y 50 donde se muestra como la calidad converge en muy pocas generaciones tras cada reinicio.

el nivel de exploración, con un valor de μ de 10 individuos, se consigue que el algoritmo resuelva todas las ejecuciones realizadas.

De la misma forma que en la sección anterior con el RCGA, en esta sección se ha caracterizado el comportamiento del CMA-ES. En un primer análisis se estudiaron los resultados obtenidos por el algoritmo ejecutado con la configuración recomendada por [Auger and Hansen, 2005]. Las conclusiones de este análisis permitieron identificar los problemas de este algoritmo con las funciones multimodales de alta dimensionalidad o unimodales de camino muy largo, en especial aquellas de tipo *L-Separable* y *NL-Separable*. El siguiente paso de la metodología de análisis propuesta en este trabajo consiste en, una vez identificados los problemas del algoritmo original, estudiar su estrategia de búsqueda en mayor detalle y tratar de identificar qué parámetros de esta estrategia pueden modificarse para tratar de solucionar estas funciones problemáticas. En el caso del CMA-ES aumentando el nivel de exploración

D	N	μ	SR	CPEM	G_{NE}
30	70	1	0.04	$3.33e{+}0 \pm 1.87e{+}0$	$1.40e{-}1 \pm 6.91e{-}2$
		10	1.00	$3.39e{-}8 \pm 9.84e{-}10$	$6.62e{-}5 \pm 4.35e{-}6$
50	10	1	0.00	$1.50e{+}1 \pm 6.90e{+}0$	$2.22e{-}1 \pm 5.24e{-}2$
		10	1.00	$2.24e{-}8 \pm 5.39e{-}10$	$5.18e{-}5 \pm 2.96e{-}6$

Tabla 5.11: Comparación de los resultados del CMA-ES para la función *Levy* incrementando el valor original de μ de 1 a 10 individuos en ambos casos.

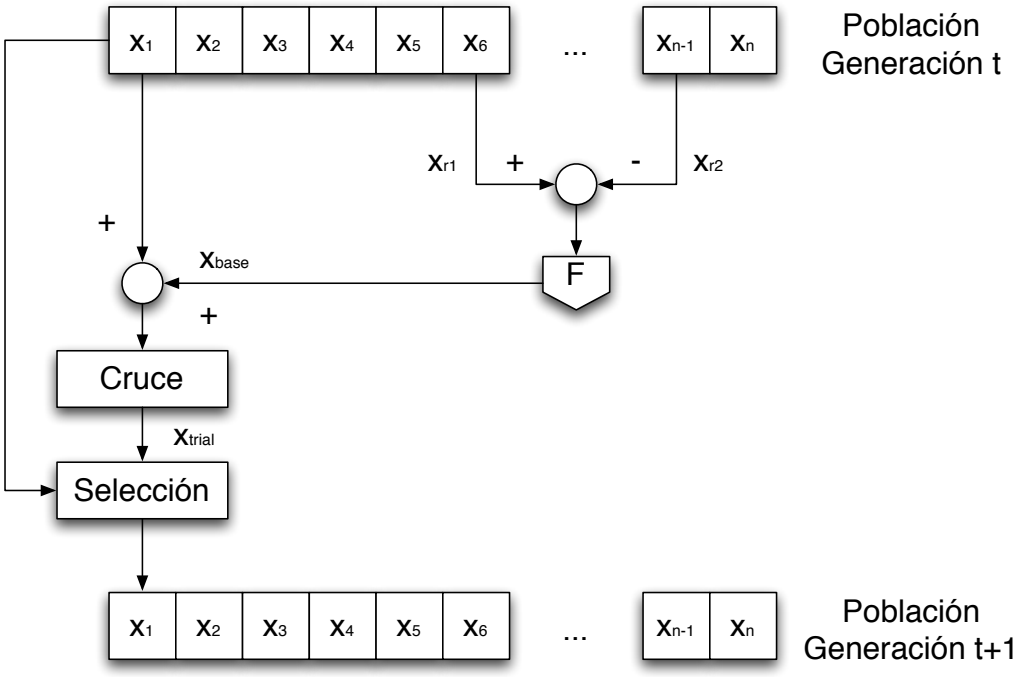


Figura 5.11: Esquema básico de funcionamiento del algoritmo DE

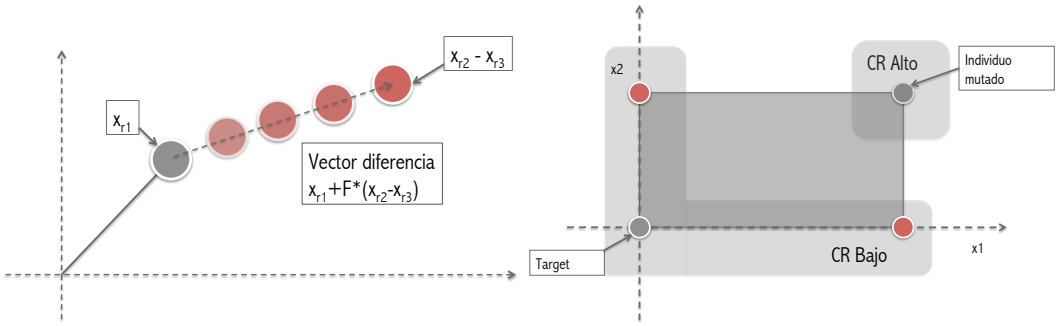
fue posible resolver todas las funciones, excepto aquellas de tipo *L-Separable*. La estrategia de modificación de la matriz de covarianzas utilizada por el algoritmo provoca que el rendimiento en este tipo de funciones sea muy pobre.

5.3 Differential Evolution (DE)

5.3.1 Descripción del algoritmo

Rainer Storn y Kenneth Price presentaron el algoritmo Differential Evolution en 1995 como un algoritmo simple basado en poblaciones para minimización de funciones [Storn and Price, 1995, Storn and Price, 1997]. En cada generación del algoritmo se aplica el operador de mutación para cada uno de los individuos de la población. Para cada vector padre, también llamado vector objetivo o vector *target*, se genera una nueva solución a partir de tres vectores: un vector llamado vector base y dos vectores escogidos de forma aleatoria. En este algoritmo la fase de reemplazo sigue una técnica voraz o *greedy*: si el vector mutado o resultante tiene mejor valor en la función de calidad que el vector objetivo, el vector mutado reemplaza a aquel con el que está siendo comparado, el vector objetivo. En la figura 5.11 se muestra un esquema de esta estrategía de búsqueda.

La característica principal de este algoritmo es que utiliza una mutación basada en la distribución de la solución dentro de la población actual. La dirección de búsqueda y la longitud del paso de búsqueda dependen de la localización de los individuos elegidos para calcular los valores de mutación. Es decir, extrayendo información sobre la distancia y la dirección entre los individuos para generar variaciones aleatorias, se obtiene como resultado un esquema adaptativo con excelentes propiedades de convergencia. Para conseguir un incremento en la diversidad de la población se aplica un operador de cruce al vector mutado [Storn and Price,



(a) Esquema del operador de mutación. (b) Esquema del operador de cruce.

Figura 5.12: Esquemas de los operadores del algoritmo DE.

1997]. Este operador de cruce consiste en elegir algunos genes del vector objetivo y otros del nuevo vector. La ejecución o no del operador de cruce está determinada por un parámetro escogido por el usuario denominado *CR*. Este parámetro controla la influencia del padre en la descendencia generada. Valores altos para el parámetro *CR* significa menor influencia del padre. El parámetro *F* es el encargado de ponderar la influencia de las soluciones escogidas para generar el valor de mutación. El comportamiento de los operadores del algoritmo DE se representa de manera esquemática en las figuras 5.12a y 5.12b.

Actualmente existen muchas variantes de este algoritmo. Para nombrar a las distintas variantes de DE se utiliza la siguiente nomenclatura *DE/x/y/z*, donde DE son las siglas de Differential Evolution, x indica la técnica utilizada para escoger el vector *base*, y indica el número de pares de soluciones que se utilizan para generar el vector diferencia y, por último, z indica el tipo de cruce que se utiliza.

Las estrategias para seleccionar el vector *base* más utilizadas son las siguientes:

- ***rand:*** el vector *base* se escoge de forma aleatoria entre todos los individuos de la población, excepto el vector *objetivo* [Storn and Price, 1997, Price et al., 2005].
- ***best:*** el vector *base* es el mejor individuo de la población.
- ***current-to-random:*** el vector *base* se sustituye por $\lambda x_{r1} + (1 - \lambda)x_i$.
- ***current-to-best:*** el vector *base* se sustituye por $\lambda x_{best} + (1 - \lambda)x_i$.

  En los dos casos anteriores, se combina información del padre para generar el vector mutado. El parámetro λ controla la influencia del mejor individuo o del individuo aleatorio sobre el vector padre.

- ***current-to-pbest:*** en la expresión anterior, el vector x_{best} se elige entre los p mejores individuos de la población [Zhang and Sanderson, 2009].

En cuanto al número de vectores diferencia, lo más usual es utilizar un sólo vector, aunque en algunas ocasiones utilizar dos vectores diferencia ha proporcionado resultados satisfactorios.

El último parámetro a ajustar en el algoritmo DE es el tipo de cruce utilizado. Actualmente existen dos esquemas de cruce [Storn and Price, 1995, Storn and Price, 1997, Price et al., 2005]:

- ***Binomial (bin):*** el cruce binomial es muy parecido al cruce uniforme utilizado en los algoritmos evolutivos. Para seleccionar aquellos genes que se cruzan se utiliza una distribución binomial, cada gen tiene una probabilidad *CR* de ser cruzado. El vector *trial* tendrá al menos un elemento del vector *mutado*.

- ***Exponencial (exp):*** este tipo de cruce es similar a las variantes de cruce por uno o dos puntos utilizados en los algoritmos genéticos. Comenzando por una posición aleatoria del vector *target*, el vector *trial* contendrá una secuencia consecutiva de componentes del vector *mutado*. Aunque este esquema de cruce fue utilizado en la primera versión del DE [Storn and Price, 1995], actualmente la mayor parte de las implementaciones utilizan el cruce binomial.

El principal problema de los esquemas de cruce del DE es que no son invariantes ante rotaciones, lo cual hace que el rendimiento del algoritmo ante funciones rotadas sea bajo. Sin embargo, si eliminamos el operador de cruce para solucionar ese problema el rendimiento del algoritmo frente a funciones multimodales decrece [Zaharie, 2009]. En este trabajo se utilizará la versión clásica del DE, es decir, el esquema *DE/rand/1/bin* descrito en [Storn and Price, 1997]. El algoritmo 5 muestra el pseudocódigo de esta versión que además está disponible en la librería JEAF (www.gii.udc.es/jeaf).

5.3.2 Caracterización del algoritmo DE

Como se comentó en la introducción del capítulo, partimos de que las dos primeras etapas del procedimiento de caracterización ya han sido llevadas a cabo y se ha seleccionado el conjunto de funciones prueba caracterizado en la sección 4.1.2 cuyas propiedades se resumen en las tablas 4.3, 4.4 y 4.5. En cuanto a las medidas de error y rendimiento, se utilizarán el CPEM, el SR y el G_{NE}. Describimos a continuación la tercera etapa del procedimiento propuesto, centrada en la ejecución y el análisis de resultados.

Ejecución de las pruebas

El primer paso consiste en ejecutar el algoritmo sobre este conjunto de prueba siguiendo las indicaciones explicadas en 4.3.1. Los parámetros propios del algoritmo DE utilizados en este trabajo se muestran en la tabla 5.12. Como parte de este tercer paso se llevó a cabo un análisis poblacional con el objetivo de determinar el tamaño poblacional óptimo para cada función del conjunto de prueba. En el siguiente apartado se analizan los resultados obtenidos en este sentido.

Análisis poblacional En la tercera columna de la tabla 5.13 se muestran los tamaños de población óptimos obtenidos por el DE. Como ya se comentó anteriormente, no se presentan en este trabajo todos los datos existentes por cuestiones de espacio, sino únicamente los que obtienen el mejor resultado final.

Analizando los tamaños de población obtenidos comenzando por las funciones de baja dimensionalidad, vemos que la mayor parte de ellas no requieren más de 20 individuos para resolver el problema. Los casos excepcionales son las funciones *Easom*, *Shekel 5*, *Shekel 7*, *Shekel 10* y *Shekel's Foxholes* en los cuales es necesaria una población de 40 o 50 individuos según los casos. La topología de estas funciones, tanto la *Easom* que es unimodal como en

Algoritmo 5 Esquema básico del algoritmo Evolución Diferencial.

Entrada:

NP, número de individuos de la población.
D, número de genes de cada cromosoma.
t_{max}, número máximo de generaciones.
CR, influencia del padre en el hijo generado.
F, peso de la suma ponderada, influencia de las soluciones en la nueva solución.

$t \leftarrow 0$
Generar la población inicial de forma aleatoria: $\mathbf{x}_t^i, \forall i, i = 1, \ldots, NP$
Evaluar $f(\mathbf{x}_t^i), \forall i, i = 1, \ldots, NP$
for $t = 1$ hasta t_{max} **do**
  for $i = 1$ hasta NP **do**
    Seleccionar de forma aleatoria: $r_1 \neq r_2 \neq r_b$
    $j_{rand} = rand(1, D)$
    for $j = 1$ hasta D **do**
      if $rand_j(0, 1] < CR$ **or** $j = j_{rand}$ **then**
        $u_{j,t+1}^i = x_{j,t}^{r_b} + F(x_{j,t}^{r_1} - x_{j,t}^{r_2})$
      else
        $u_{j,t+1}^i = x_{j,t}^i$
      end if
    end for
    if $f(\mathbf{u}_{t+1}^i) \leq \mathbf{x}_t^i$ **then**
      $\mathbf{x}_{t+1}^i = \mathbf{u}_{t+1}^i$
    else
      $\mathbf{x}_{t+1}^i = \mathbf{x}_t^i$
    end if
  end for
  $t \leftarrow t + 1$
end for

los otros cuatro casos que son multimodales, provoca la necesidad de un mayor nivel de exploración por parte del algoritmo. Esto se consigue aumentando el tamaño de la población.

En las funciones de alta dimensionalidad, igual que se hizo en el caso del CMA-ES, se distingue entre funciones unimodales y multimodales. En el primer caso, las funciones unimodales, no es necesario un tamaño de población elevado para resolverlas excepto en el caso de la función *Perm*, que debido a su dificultad requiere un mayor número de individuos aunque en ninguno de los casos analizados (tamaños de población de 10 a 100 individuos) se resuelve.

En el caso de las funciones multimodales, el tamaño poblacional óptimo es de 20 individuos excepto en las funciones *Rastrigin*, *Schwefel*, *Griewank* de dimensión 10 y *Rosenbrock*. Las tres primeras funciones comparten la característica de presentar un elevado número de óptimos, en concreto 11^n, 8^n y 4^n respectivamente. Además son las funciones con mayor distancia entre centros de atracción $4.86e{-}1$, $9.20e{-}1$ y $4.55e{-}1$. Por este motivo son las funciones que mayor nivel de exploración requieren. Una de las estrategias para aumentar el nivel de exploración es utilizar un tamaño de población alto. En el caso de la función *Rosenbrock*, a pesar de tener solamente dos óptimos locales como se indica en la tabla 4.4 de la sección 4.1.3, su topología provoca que el algoritmo tenga tendencia a caer en el óptimo

Operador	Parámetro	Valor
Mutación	Estrategia *base*	*rand*
	Núm. de vectores	1
	F	0.9
Cruce	Tipo	*bin*
	CR	0.1 o 0.9, dependiendo del tipo de función.

Tabla 5.12: Parámetros de configuración utilizados para la caracterización del DE

local y que requiera, al igual que las funciones anteriores, un mayor tamaño de población para aumentar el nivel de exploración y resolver la función.

Resumiendo, generalmente el número de individuos requeridos por el DE para resolver una función del conjunto no es mayor de 30. Si la topología de la función requiere un mayor nivel de exploración, como es el caso de las funciones multimodales con un número elevado de centros de atracción o con amplia distancia entre ellos, entonces es necesario aumentar el tamaño de la población para resolverla.

Análisis de resultados

Como en las secciones anteriores, la caracterización del DE se ha realizado ejecutando el algoritmo sobre las funciones del conjunto de prueba utilizando el procedimiento que se explica en la sección 4.3.1. Los resultados obtenidos se muestran en la tabla 5.13.

Tabla 5.13: Resultados obtenidos por el algoritmo DE en el conjunto de funciones utilizadas en este trabajo.

Función	D	N	SR	CPEM	GNE
Axis Parallel	10	20	1.00	$7.34e{-8} \pm 2.55e{-9}$	$1.78e{-6} \pm 3.34e{-7}$
	30	20	1.00	$8.98e{-8} \pm 1.58e{-9}$	$5.83e{-7} \pm 1.11e{-7}$
	50	20	1.00	$9.88e{-8} \pm 1.61e{-9}$	$2.90e{-7} \pm 1.05e{-7}$
Schwefel 2.22	10	20	1.00	$9.19e{-8} \pm 2.38e{-9}$	$1.41e{-8} \pm 2.42e{-9}$
	30	20	1.00	$1.01e{-7} \pm 1.44e{-9}$	$4.49e{-9} \pm 3.92e{-10}$
	50	20	1.00	$1.08e{-7} \pm 1.50e{-9}$	$2.75e{-9} \pm 1.57e{-10}$
Sphere Model	10	20	1.00	$6.90e{-8} \pm 1.73e{-9}$	$3.24e{-6} \pm 2.00e{-7}$
	30	20	1.00	$8.08e{-8} \pm 1.60e{-9}$	$1.83e{-6} \pm 4.38e{-8}$
	50	20	1.00	$8.92e{-8} \pm 1.42e{-9}$	$1.41e{-6} \pm 3.23e{-8}$
Step	10	10	1.00	$1.47e{-8} \pm 1.28e{-9}$	$3.18e{-3} \pm 5.73e{-4}$
	30	20	1.00	$3.51e{-8} \pm 1.40e{-9}$	$2.76e{-3} \pm 2.59e{-4}$
	50	20	1.00	$3.88e{-8} \pm 9.96e{-10}$	$2.75e{-3} \pm 2.34e{-4}$
SumOf	10	10	1.00	$1.69e{-8} \pm 2.33e{-9}$	$6.57e{-2} \pm 1.63e{-2}$
	30	20	1.00	$2.13e{-8} \pm 1.34e{-9}$	$2.19e{-1} \pm 2.61e{-2}$
	50	20	1.00	$1.97e{-8} \pm 1.92e{-9}$	$2.69e{-1} \pm 2.36e{-2}$
Easom	2	40	1.00	$2.72e{-7} \pm 7.50e{-8}$	$7.67e{-4} \pm 3.80e{-3}$
Schwefel 2.21	10	10	1.00	$3.14e{-7} \pm 8.01e{-9}$	$7.07e{-8} \pm 3.23e{-7}$
	30	20	1.00	$5.82e{-7} \pm 1.19e{-8}$	$5.91e{-9} \pm 4.89e{-10}$
	50	20	1.00	$7.57e{-7} \pm 1.28e{-8}$	$5.59e{-9} \pm 3.35e{-10}$
Colville	4	20	1.00	$1.13e{-7} \pm 2.88e{-8}$	$3.63e{-3} \pm 1.21e{-2}$
Matyas	2	20	1.00	$3.86e{-8} \pm 5.16e{-9}$	$5.37e{-4} \pm 8.61e{-4}$
Perm	10	60	0.72	$1.56e{-6} \pm 2.94e{-6}$	$1.12e{-2} \pm 3.25e{-3}$
	30	70	0.00	$3.74e{+0} \pm 8.87e{+0}$	$4.65e{-3} \pm 2.60e{-3}$
	50	100	0.00	$5.29e{+99} \pm 2.59{+100}$	$6.23e{-2} \pm 1.75e{-2}$

Tabla 5.13 -- continua desde la página anterior

Función	D	N	SR	CPEM	GNE
Schwefel 1.2	10	20	1.00	$1.44e{-}7 \pm 1.78e{-}8$	$4.75e{-}6 \pm 6.72e{-}7$
	30	20	1.00	$4.73e{-}7 \pm 2.85e{-}8$	$3.09e{-}6 \pm 1.09e{-}7$
	50	20	0.96	$2.26e{-}6 \pm 7.02e{-}6$	$3.00e{-}6 \pm 2.68e{-}6$
Zakharov	10	20	1.00	$1.12e{-}7 \pm 1.52e{-}8$	$4.13e{-}5 \pm 3.40e{-}6$
	30	30	1.00	$5.47e{-}7 \pm 4.02e{-}8$	$2.39e{-}5 \pm 4.98e{-}7$
	50	30	0.00	$1.06e{+}0 \pm 5.19e{+}0$	$3.97e{-}3 \pm 1.94e{-}2$
Aluffi-Pentini's	2	20	1.00	$3.94e{-}8 \pm 4.44e{-}9$	$1.16e{-}4 \pm 5.85e{-}5$
Becker and Lago	2	20	1.00	$5.06e{-}8 \pm 7.27e{-}9$	$7.39e{-}1 \pm 3.12e{-}1$
Bohachevsky 1	2	20	1.00	$5.66e{-}8 \pm 5.63e{-}9$	$6.26e{-}6 \pm 7.61e{-}6$
Cosine Mixture	10	20	1.00	$4.98e{-}8 \pm 1.91e{-}9$	$8.55e{-}5 \pm 1.11e{-}5$
	30	20	1.00	$5.90e{-}8 \pm 1.24e{-}9$	$5.11e{-}5 \pm 1.83e{-}6$
	50	20	1.00	$6.47e{-}8 \pm 1.07e{-}9$	$3.87e{-}5 \pm 1.00e{-}6$
Rastrigin	10	30	1.00	$1.18e{-}7 \pm 3.49e{-}9$	$4.61e{-}6 \pm 5.83e{-}7$
	30	30	1.00	$1.59e{-}7 \pm 2.79e{-}9$	$2.56e{-}6 \pm 5.81e{-}8$
	50	50	1.00	$3.78e{-}7 \pm 4.44e{-}9$	$1.98e{-}6 \pm 6.42e{-}8$
Schwefel	10	40	1.00	$1.66e{-}7 \pm 5.19e{-}9$	$2.38e{-}6 \pm 3.38e{-}7$
	30	60	1.00	$3.27e{-}7 \pm 5.99e{-}9$	$1.82e{-}6 \pm 7.96e{-}8$
	50	80	1.00	$5.70e{-}7 \pm 7.53e{-}9$	$1.72e{-}6 \pm 2.99e{-}8$
Ackleys	10	20	1.00	$1.10e{-}7 \pm 3.43e{-}9$	$7.78e{-}9 \pm 7.81e{-}10$
	30	20	1.00	$1.01e{-}7 \pm 6.26e{-}8$	$7.93e{-}9 \pm 5.09e{-}10$
	50	20	1.00	$1.49e{-}7 \pm 8.97e{-}8$	$7.76e{-}9 \pm 1.20e{-}10$
Griewank	30	20	1.00	$6.18e{-}8 \pm 1.36e{-}9$	$1.67e{-}6 \pm 1.31e{-}7$
	50	20	1.00	$9.29e{-}8 \pm 3.40e{-}9$	$1.70e{-}6 \pm 7.03e{-}8$
Levy	10	20	1.00	$5.03e{-}8 \pm 2.38e{-}9$	$1.30e{-}4 \pm 2.01e{-}5$
	30	20	1.00	$6.18e{-}8 \pm 1.36e{-}9$	$7.08e{-}5 \pm 2.85e{-}6$
	50	20	1.00	$6.95e{-}8 \pm 1.16e{-}9$	$5.65e{-}5 \pm 1.96e{-}6$
Penalized 1	10	20	1.00	$6.00e{-}8 \pm 2.73e{-}9$	$4.42e{-}5 \pm 3.56e{-}6$
	30	20	1.00	$6.98e{-}8 \pm 1.56e{-}9$	$4.45e{-}5 \pm 1.37e{-}6$
	50	20	1.00	$7.76e{-}8 \pm 1.68e{-}9$	$4.46e{-}5 \pm 1.46e{-}6$
Penalized 2	10	20	1.00	$6.27e{-}8 \pm 2.80e{-}9$	$1.89e{-}5 \pm 2.81e{-}6$
	30	30	1.00	$1.21e{-}7 \pm 2.51e{-}9$	$1.14e{-}5 \pm 6.83e{-}7$
	50	20	1.00	$8.51e{-}8 \pm 1.73e{-}9$	$8.83e{-}6 \pm 3.17e{-}7$
Beale	2	20	1.00	$4.70e{-}8 \pm 5.48e{-}9$	$3.85e{-}4 \pm 3.553{-}4$
Bohachevsky 2	2	20	1.00	$6.26e{-}8 \pm 6.35e{-}9$	$5.83e{-}6 \pm 7.94e{-}6$
Dekkers and Aarts	2	10	1.00	$1.85e{-}8 \pm 5.29e{-}9$	$5.77e{-}1 \pm 5.20e{-}1$
Goldstein Price	2	20	1.00	$5.25e{-}8 \pm 2.15e{-}4$	$6.25e{-}5 \pm 1.05e{-}4$
Griewank	10	50	1.00	$3.93e{-}7 \pm 3.26e{-}8$	$1.40e{-}6 \pm 2.38e{-}7$
Hartman 3	3	20	1.00	$2.94e{-}8 \pm 4.17e{-}9$	$5.47e{-}3 \pm 2.62e{-}3$
Hartman 6	6	20	0.28	$1.06e{-}1 \pm 1.99e{-}1$	$7.58e{-}1 \pm 3.34e{-}1$
Rosenbrock	10	50	1.00	$6.47e{-}7 \pm 3.32e{-}8$	$5.37e{-}6 \pm 3.60e{-}6$
	30	40	0.24	$8.48e{-}1 \pm 1.44e{+}0$	$3.78e{-}3 \pm 4.94e{-}3$
	50	30	0.00	$3.67e{+}1 \pm 3.35e{+}1$	$4.29e{-}2 \pm 2.79e{-}2$
Kowalik's	4	60	0.68	$3.14e{-}4 \pm 4.61e{-}4$	$2.50e{-}1 \pm 3.61e{-}1$
Shekel Family 5	4	50	1.00	$2.07e{-}7 \pm 1.78e{-}8$	$3.02e{-}5 \pm 8.62e{-}6$
Shekel Family 7	4	40	1.00	$1.35e{-}7 \pm 1.28e{-}8$	$1.44e{-}4 \pm 4.30e{-}5$
Shekel Family 10	4	50	1.00	$1.72e{-}7 \pm 1.22e{-}8$	$1.49e{-}5 \pm 4.92e{-}5$
Shekel's Foxholes	2	40	1.00	$1.17e{-}7 \pm 2.36e{-}8$	$1.01e{-}2 \pm 3.89e{-}2$
SixHump Camel Back	2	20	1.00	$5.02e{-}8 \pm 1.74e{-}9$	$1.00e{-}1 \pm 9.82e - 2$

Configuración básica En primer lugar se analizarán los resultados obtenidos por el algoritmo DE utilizando la configuración propuesta por los autores en [Storn and Price, 1997]. Observando los resultados obtenidos tras la ejecución del algoritmo, se concluye que el DE

tiene problemas con las funciones no separables de alta dimensionalidad (dimensiones 30 y 50). El algoritmo falla en cuatro funciones en las cuales aparecen de manera conjunta estas dos características: las funciones *Perm*, *Zakharov* y *Rosenbrock*. El algoritmo también presenta problemas con la función *Hartman 6* (no separable de dimensión 6). Este resultado llama la atención porque la función no es de alta dimensionalidad y el DE resuelve todas las funciones no separables de baja dimensionalidad excepto esta. Este comportamiento será discutido y explicado más adelante.

El DE resuelve todas las funciones *L-Separables* y *NL-Separables*. Este comportamiento es consecuencia de su operador de cruce, el cual está controlado por el parámetro *CR* que determina cuántos genes mutados pasan de una generación a la siguiente controlando la dirección de búsqueda del algoritmo. La probabilidad de que un gen cambie puede calcularse a partir del tipo de cruce utilizado, la dimensión del problema y el valor de CR (ver [Zaharie, 2009] para una explicación detallada). Una probabilidad de cambio baja puede conseguirse utilizando un valor bajo de CR. En este caso se utilizó un valor de 0.1 como se recomienda en [Rönkkönen et al., 2005], permitiendo que el proceso de optimización busque en direcciones ortogonales y, como consecuencia, se optimice de forma independiente cada uno de los parámetros del genotipo. Por lo tanto, el DE es una buena opción para resolver funciones separables con esta configuración de sus operadores.

En las funciones *No-Separables*, el comportamiento del DE es exitoso en funciones de baja dimensionalidad (excepto en la función *Hartman 6*) y su rendimiento empeora en todas las funciones de alta dimensionalidad con más de 10 dimensiones. Normalmente, los problemas aparecen con dimensiones 30 y 50, aunque hay algunos fallos en la función *Perm* de 10 dimensiones (con una SR del 72%). Esta función tiene un camino hasta el óptimo muy largo como ya se ha comentado al analizar otros algoritmos. De entre las restantes funciones no separables de alta dimensionalidad, la función *Schwefel 1.2* no representa un problema para el DEy simplemente tiene algunos fallos en dimensión 50 (SR 96%). El algoritmo no es capaz de resolver la función *Zakharov* de dimensión 50 (SR 0%). En la función *Rosenbrock*, que es multimodal en vez de unimodal como los tres casos anteriores, el DE falla tanto en dimensión 30 (SR 24%) como en dimensión 50 (SR 0%). Como ya ha sido comentado, la mejor estrategia para resolver funciones no separables es aprender las dependencias entre los parámetros. En el caso del algoritmo DE, el parámetro CR, que controla el número de nuevos genes que pasan a la siguiente generación, debe ser incrementado para obtener el comportamiento contrario al que se mostró para funciones separables [Zaharie, 2009]. De hecho, en los resultados mostrados en la tabla 5.13, el valor de CR se incrementó a 0.9 para las ejecuciones correspondientes a funciones no separables, lo cual mejoró los resultados obtenidos inicialmente. Sin embargo, esto no es suficiente para resolver los casos mencionados.

Existen dos funciones de tipo *No-Separables* de baja dimensionalidad que se deben comentar: las fuciones *Kowaliks* y *Hartman 6*. El DE falla algunas ejecuciones de la función *Kowaliks* obteniendo un SR del 60%, esta función tiene 4 dimensiones. Es una función que presenta muchas dificultades debido a su topología, el centro de atracción óptimos no es el de mayor tamaño (esta no es la única función donde esta característica representa un problema para el DE). Como se muestra en la tabla 4.4, esta función tiene 65 centros de atracción y el tamaño de los mismos crece a medida que los individuos se alejan del óptimo global. Por lo tanto, la función es deceptiva y el DE tenderá a converger hacia los centros de atracción de mayor tamaño. Finalmente, en la función *Hartman 6* de 6 dimensiones el DE obtiene un SR del 28%. En el conjunto de funciones de prueba hay tres funciones *No-Separables* multimodales de 6 o más dimensiones: la función *Griewank* (10D), la función *Rosenbrock* (10D, 30D and 50D) y la función *Hartman 6* (6D). El DE no tiene problemas con la función *Griewank*, el

rendimiento del algoritmo en la función *Rosenbrock* comienza a empeorar a partir de dimensión 30 y la función *Hartman 6* raramente se resuelve. La clave es el tipo de multimodalidad: la función *Hartaman 6* presenta dos centros de atracción aislados y la función *Rosenbrock* presenta nueve, dos de ellos ocupan casi la totalidad del espacio de calidad considerándose los centros de atracción principales de dicha función. Por otro lado, la función *Griewank* tiene varios centros de atracción repartidos por el espacio de calidad, esta distribución facilita el proceso de exploración del DE. La función *Hartman 6* es más difícil que la función *Rosenbrock* debido a que en la primera el centro de atracción del óptimo es más pequeño que el otro.

Para terminar con el análisis del algoritmo DE, a partir de los resultados obtenidos se pueden concluir las siguientes características:

- Su rendimiento es satisfactorio en funciones separables debido a su capacidad de buscar en direcciones ortogonales utilizando un valor bajo de CR.
- Presenta problemas con funciones *No-Separables* de alta dimensionalidad, básicamente porque no tiene en cuenta las dependencias entre parámetros, aunque su rendimiento mejora con valores altos de CR. Este comportamiento es independiente de la modalidad del espacio de calidad.
- En funciones unimodales, un camino muy largo hacia el óptimo empeora el rendimiento del DE, pero en menor medida que otros algoritmos ya que solamente ocurre en la función *Perm* que también es *No-Separable* de alta dimensionalidad. Por tanto, el camino largo es un problema añadido (el DE tiene menos dificultades con las otras funciones de alta dimensionalidad, *No-Separables* y unimodales) pero, probablemente, no sea el problema principal de la función *Perm*. El comportamiento del DE en las funciones unimodales es satisfactorio debido a que su estrategia de búsqueda permite un ajuste automático del paso de mutación basado en la distribución de la población sobre el espacio de calidad durante el proceso. En las generaciones iniciales del DE, la población está repartida sobre el espacio de búsqueda y, como el paso de mutación depende de la distancia entre individuos, su valor es alto. A medida que la población converge, la distancia decrece y, en consecuencia, también lo hace el paso de mutación.
- En funciones *No-Separables* multimodales, un número bajo de centros de atracción parece suficiente para que el número de dimensiones necesarias para observar un mal comportamiento en funciones *No-Separables* sea menor. En este caso concreto el rendimiento del algoritmo empeora en funciones con seis o más dimensiones.

Variaciones sobre la configuración básica Los resultados y las conclusiones que han sido explicados en la sección anterior fueron obtenidos utilizando la configuración recomendada en [Rönkkönen et al., 2005]. Esta configuración básica ha resultado ser muy adecuada para resolver funciones que generan espacios de calidad con diferentes características topológicas. Sin embargo, dentro del conjunto de funciones de prueba utilizado en este trabajo, existen varias funciones que no se resuelven utilizando esta configuración. Debemos pues analizar por qué motivo no se resuelven y modificar la configuración del algoritmo para tratar de arreglarlo.

En este caso las funciones no resueltas son las siguientes:

- Dos funciones de tipo unimodal y *No-Separables*: las funciones *Perm* y *Zakharov*. La función *Perm* es la que presenta el camino hasta el óptimo más largo dentro de todas las

funciones analizadas en este trabajo. En el caso de la función *Zakharov* el camino al óptimo no es excesivamente largo, situandose en la media de longitudes de las funciones consideradas en este trabajo.

- Tres funciones de tipo multimodal y, también, *No-Separables*: la función *Kowaliks*, la función *Hartman 6* y la función *Rosenbrock*. En las dos primeras, el centro de atracción óptimo no es el de mayor tamaño. No ocurre lo mismo en la función *Rosenbrock*.

Analizando las características de estas cinco funciones, todas ellas comparten el hecho de ser no separables. Como ya ha sido mencionado en esta sección, la estrategia que se debe utilizar con el DE en el caso de tratar de resolver una función no separable es establecer un valor de *CR* elevado (generalmente, 0.9) para que la búsqueda se realice en direcciones diagonales. Sin embargo, como se ha visto, esto no es suficiente para tratar con este tipo de funciones. En las funciones no separables existen dependencias entre parámetros e, idealmente, para resolver este tipo de funciones, un algoritmo debería tratar de aprender dichas dependencias durante el proceso de optimización. En el caso del DE, su estrategia de búsqueda adapta las variables del problema teniendo en cuenta dichas dependencias. Por lo tanto, para mejorar los resultados obtenidos en las cuatro funciones mencionadas, es necesario seguir otro tipo de estrategia que no involucre el operador de cruce, ya que la configuración utilizada para este operador es la óptima.

En cuanto a la modalidad de las funciones, dos de ellas son unimodales (la función *Perm* y la función *Zakharov*). En el caso de la función *Perm* de dimensión 10, el valor de CPEM que se obtiene ($1.56e-6$) indica que las ejecuciones no resueltas casi alcanzan el valor considerado como óptimo (en este trabajo 10^{-6}). En las ejecuciones con 30 y 50 parámetros, los resultados que obtiene el algoritmo están alejados del valor óptimo (un valor medio de CPEM de $3.74e{+}0$ para dimensión 30 y $5.29e{+}99$ para dimensión 50). El algoritmo, en estos dos casos, presenta una velocidad de convergencia lenta. El único caso problemático de la función *Zakharov* es en dimensión 50. De las 25 ejecuciones realizadas, 24 obtienen un valor casi óptimo con valores de CPEM entre 10^{-4} y 10^{-6} y en una de ellas la población del algoritmo converge hacia un valor de CPEM de 26.48. Con estos resultados podemos concluir que el problema del DE en este tipo de funciones es que la velocidad de convergencia es lenta y que aumentando esta velocidad podríamos obtener mejores resultados. Para aumentar la velocidad de convergencia, es necesario aumentar la presión selectiva. De hecho, la presión selectiva del DE es muy baja. Utilizando la estrategia de mutación *rand* la probabilidad de escoger el mejor de los individuos de la población cada vez que se genera una nueva solución es muy baja, concretamente, $1/(N \cdot (N-1) \cdot (N-2))$ siendo N el número de individuos de la población. Por lo tanto, existen pocas probabilidades de que la información del mejor individuo pase a la siguiente generación disminuyendo notablemente la presión selectiva. Cabe destacar que, a pesar de esta baja presión selectiva, el alto nivel de "voracidad" que presenta el algoritmo compensa en parte este problema. Para aumentar esta probabilidad y, en consecuencia, aumentar la presión selectiva se debe utilizar otra estrategia de mutación. En la bibliografía encontramos diferentes opciones. Entre las más conocidas están las estrategias *best* [Pahner and Hameyer, 2002] y *current-to-best* [Mezura-Montes et al., 2006]. Estas estrategias provocan que el comportamiento del DE sea más voraz, aumentando la velocidad de convergencia, pero pueden provocar convergencia prematura. Otra estrategia interesante es la *current-to-pbest* [Zhang and Sanderson, 2009], que permite controlar el nivel de presión selectiva deseado ajustando el parámetro p. Utilizando esta estrategia los descendientes se generan utilizando la siguiente ecuación:

$$\mathbf{v}_{i,g} = \mathbf{x}_{i,g} + F \cdot (\mathbf{x}^{p}_{best,g} - \mathbf{x}_{i,g}) + F \cdot (\mathbf{x}_{r1,g} - \mathbf{x}_{r2,g}) \tag{5.5}$$

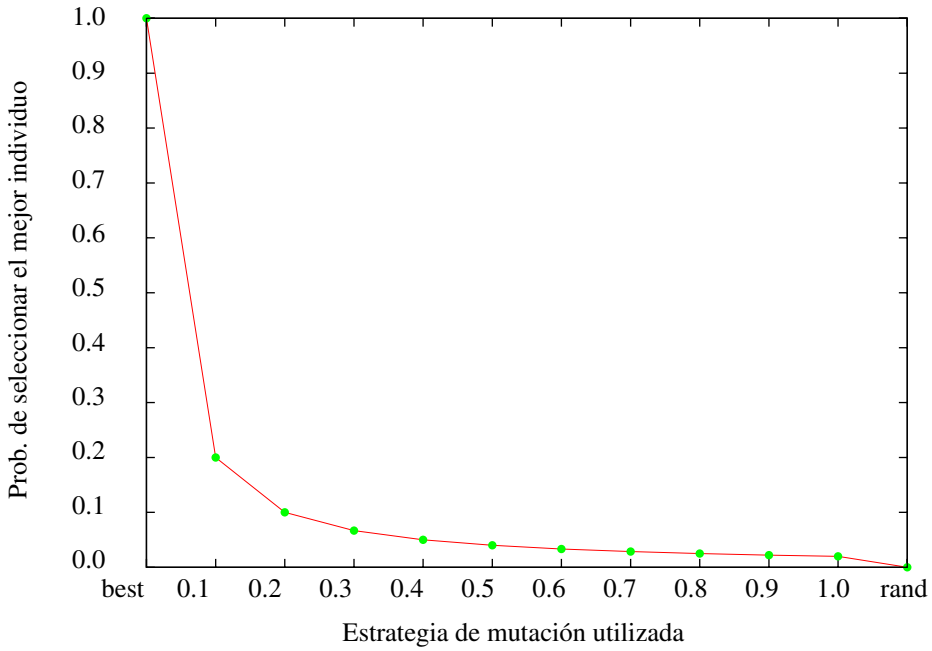


Figura 5.13: Evolución de la probabilidad de seleccionar el mejor individuo según la estrategia de mutación utilizada utilizando una población de 50 individuos. Los valores extremos se corresponden con la estrategia *best* y *rand*. Los valores intermedios se corresponden con diferentes valores del parámetro p de la estrategia *current-to-pbest*

.

donde $\mathbf{x}^{p}_{best,g}$ es un individuo seleccionado aleatoriamente entre los $100 \cdot p\%$ mejores individuos de la población (p puede tomar valores en el intervalo $(0, 1]$). Utilizando valores cercanos a 0 se obtiene una estrategia más voraz con un comportamiento similar a la estrategia *best*. Por el contrario, valores cercanos a 1 provocan que el comportamiento de la búsqueda sea más explorador, parecido a la estrategia *rand*. En la figura 5.13 se representa la probabilidad de que se utilice al mejor individuo de la población para generar la descendencia. En los extremos se encuentran la estrategia *best*, con probabilidad 1, ya que el mejor individuo siempre se utiliza para generar la nueva descendencia, y la estrategia *rand*, donde todos los individuos tienen la misma probabilidad de ser seleccionados y, por lo tanto, la probabilidad de seleccionar el mejor es baja. Los valores intermedios se corresponden con la probabilidad de seleccionar al mejor individuo utilizando la estrategia *current-to-pbest* con diferentes valores del parámetro p, la probabilidad disminuye a medida que crece el valor de p. Para tratar de mejorar los resultados obtenidos por la estrategia *rand* en las funciones *Perm* y *Zakharov* se tratará de aumentar la presión selectiva del algoritmo utilizando la estrategia *current-to-pbest*.

En la tabla 5.14 se muestran los resultados obtenidos por la función *Perm* en dimensiones 30 y 50 utilizando la estrategia *rand* de la configuración original y las estrategias *best* y *current-to-pbest* con el objetivo de tratar de aumentar la presión selectiva y, de esta forma, acelerar la velocidad de convergencia. Como se puede observar en la tabla, los mejores resultados se obtienen utilizando la estrategia *current-to-pbest*. En ninguno de los dos casos se consigue aumentar la SR, sin embargo, tanto el CPEM como el G_{NE} mejoran sus resultados. Los resultados que se obtienen utilizando la estrategia de mutación *best* demuestran que esta es una estrategia demasiado voraz y que utilizándola el DE pierde casi por completo su capacidad de exploración. Esta gran capacidad de exploración es una de las claves de su éxito.

Función	D	N	Estrategia	SR	CPEM	G_{NE}
Perm	30	70	*rand*	0.00	$3.74e{+}0 \pm 8.87e{+}0$	$4.65e{-}3 \pm 2.60e{-}3$
			best	0.00	$3.29e{+}0 \pm 5.78e{+}0$	$5.36e{-}3 \pm 2.81e{-}3$
			current-to-pbest (p = 0.20)	0.00	$5.92e{-}1 \pm 1.18e{+}0$	$3.98e{-}3 \pm 2.44e{-}3$
	50	100	*rand*	0.00	$5.29e{+}99 \pm 2.59e{+}100$	$6.23e{-}2 \pm 1.75e{-}2$
			best	0.00	$1.58e{+}168 \pm Inf.$	$1.10e{-}1 \pm 1.23e{-}1$
			current-to-pbest (p = 0.10)	0.00	$7.65e{+}0 \pm 1.66e{+}1$	$2.43e{-}3 \pm 1.08e{-}3$
Zakharov	50	30	*rand*	0.00	$1.06e{+}0 \pm 5.19e{+}0$	$3.97e{-}3 \pm 1.94e{-}2$
			best	0.00	$1.74e{+}2 \pm 1.09e{+}2$	$2.33e{-}1 \pm 8.73e{-}2$
			current-to-pbest (p = 0.10)	1.00	$7.26e{-}7 \pm 4.38e{-}8$	$1.85e{-}5 \pm 2.17e{-}7$

Tabla 5.14: Comparación de los resultados del DE en las funciones unimodales utilizando diferentes estrategias de mutación.

Para la función *Zakharov* se han realizado las mismas pruebas que las presentadas para la función *Perm* tratando de mejorar los resultados obtenidos, en este caso únicamente para dimensión 50. Los resultados obtenidos se muestran en la misma tabla. De nuevo, la estrategia de mutación *best* resulta ser demasiado voraz provocando que la población converja rápidamente y se estanque en valores lejanos al óptimo. Utilizando la estrategia *current-to-pbest* con un valor de p de 0.10, lo cual provoca un aumento de la presión selectiva, se consigue una SR del 100%, aunque el algoritmo sigue siendo más lento que el CMA-ES.

Las funciones restantes, la función *Kowaliks*, la función *Hartman 6* y la función *Rosenbrock*, son todas multimodales. En dos de los casos, la función *Kowaliks* y la función *Hartman 6*, el centro de atracción óptimo no es el de mayor tamaño y, analizando los resultados obtenidos en las ejecuciones no resueltas, el algoritmo tiene tendencia a caer en aquel que no es el óptimo. En el tercer caso, la función *Rosenbrock*, la amplitud del centro de atracción no óptimo provoca el mismo comportamiento que en los dos casos anteriores. El problema que presenta el DE en este tipo de funciones surge de su poca capacidad de explotación. En los tres casos, la estrategia exploradora del algoritmo provoca que la población se distribuya sobre el espacio de búsqueda y que la mayor parte de los individuos se sitúen en los centros de atracción no óptimos. De forma que, cuando comienza la fase explotadora del algoritmo, este concentre la búsqueda en esos centros de atracción convergiendo en óptimos locales. Para tratar de mejorar los resultados proporcionados por la configuración tradicional del algoritmo se han utilizado, de nuevo, las estrategias *best* y *current-to-pbest* que ayudan a aumentar la capacidad de explotación del algoritmo. Los resultados obtenidos tras ejecutar estas versiones del DE sobre cada una de las funciones analizadas se presentan en la tabla 5.15

Como demuestran los resultados obtenidos para la función *Kowaliks*, si se aumenta la capacidad de explotación utilizando la estrategia *current-to-pbest* con un valor de p de 0.50, se obtiene una SR del 96%, es decir, únicamente una ejecución de las 25 realizadas no es capaz de resolver la función. Los resultados obtenidos por la función *Hartman 6*, demuestran que aumentando la presión selectiva se consigue mejorar los resultados obtenidos por la estrategia *rand*, incrementando en un 20% la SR. Para finalizar, analizando los resultados obtenidos por la función *Rosenbrock* con las diferentes estrategias de mutación, es posible concluir que de nuevo, la estrategia *best* es demasiado voraz, provocando la convergencia prematura de la población hacia el óptimo local. En cambio, la estrategia *current-to-pbest* obtiene los mejores resultados, mejorando tanto los valores de SR como lo valores de CPEM y G_{NE}.

Función	D	N	Estrategia	SR	CPEM	G_{NE}
Kowaliks	4	60	*rand*	0.68	$3.14e{-4} \pm 4.61e{-4}$	$2.50e{-1} \pm 3.61e{-1}$
			best	0.44	$4.36e{-3} \pm 7.86e{-3}$	$3.93e{-1} \pm 3.54e{-1}$
			current-to-pbest (p = 0.50)	0.96	$3.67e{-5} \pm 1.79e{-4}$	$3.28e{-2} \pm 1.50e{-1}$
Hartman 6	6	20	*rand*	0.28	$1.06e{-1} \pm 1.99e{-1}$	$7.58e{-1} \pm 3.34e{-1}$
			best	0.48	$8.72e{-2} \pm 1.13e{-1}$	$4.48e{-1} \pm 4.59e{-1}$
			current-to-pbest (p = 0.10)	0.48	$6.58e{-2} \pm 6.32e{-2}$	$4.72e{-1} \pm 4.46e{-1}$
Rosenbrock	30	40	*rand*	0.24	$8.48e{-1} \pm 1.44e{+0}$	$3.78e{-3} \pm 4.94e{-3}$
			best	0.12	$1.22e{+2} \pm 5.91e{+2}$	$1.25e{-2} \pm 3.48e{-2}$
			current-to-pbest (p = 0.05)	0.92	$3.19e{-1} \pm 1.08e{+0}$	$9.74e{-4} \pm 3.29e{-3}$
	50	30	*rand*	0.00	$3.67e{+1} \pm 3.35e{+1}$	$4.29e{-2} \pm 2.79e{-2}$
			best	0.00	$5.03e{+1} \pm 1.08e{+2}$	$2.23e{-2} \pm 2.70e{-2}$
			current-to-pbest (p = 0.10)	0.04	$2.88e{+0} \pm 4.26e{+0}$	$5.58e{-3} \pm 6.15e{-3}$

Tabla 5.15: Comparación de los resultados del DE para las funciones multimodales utilizando diferentes estrategias de mutación.

Resumiendo, la estrategia de búsqueda del algoritmo DE utilizado en este trabajo y cuya configuración es la más recomendada, tiende a seguir una estrategia exploradora basándose en la distribución de la población en el espacio de calidad. Esta estrategia es la clave de su éxito, sin embargo, como se ha visto, existen funciones en las que esta estrategia no es adecuada. Estos casos se corresponden con funciones unimodales de camino muy largo en las cuales es necesaria una velocidad de convergencia mayor que la que tiene la estrategia *DE/rand/1/bin* y con funciones multimodales con pocos centros de atracción en las que el algoritmo tiende a converger hacia el centro de atracción no óptimo, donde es necesaria un mayor nivel de explotación. En el algoritmo DE estas propiedades están relacionadas y dependen de la estrategia de mutación utilizada. En estos casos es necesaria una estrategia de mutación más voraz que la estrategia *rand* pero con una menor presión selectiva que la estrategia *best*. En este trabajo se ha optado por utilizar una estrategia *current-to-pbest* que, a través del parámetro p, permite ajustar la presión selectiva y, como consecuencia, el balance entre exploración y explotación, mejorando los resultados de las cuatro funciones problemáticas para la estrategia básica. En el caso de las funciones unimodales, utilizando un valor de p de 0.10, se obtiene el balance exploración / explotación óptimo para resolver este tipo de funciones. En el caso de las funciones multimodales, la elección del valor de p depende de la topología de la función y, como se ha visto en este apartado, concretamente del número de óptimos locales. A mayor número de óptimos locales (véase función *Kowaliks*), es necesaria una estrategia con un nivel de exploración más alto que de explotación y, por lo tanto, con un valor de p mayor.

El procedimiento de caracterización desarrollado en este trabajo ha sido aplicado sobre el algoritmo DE con el objetivo de caracterizar su comportamiento ante un conjunto de funciones prueba con diversas características. El primer análisis se realizó utilizando los parámetros de configuración recomendados en [Storn and Price, 1997]. Tras este primer análisis es posible concluir las características de las funciones que provocan un rendimiento pobre en el algoritmo. Estas funciones son, principalmente, las funciones de tipo *No-Separable* que, además, sean unimodales de camino largo o multimodales cuyo centro de atracción óptimo

no sea el de mayor tamaño. El segundo análisis sobre el algoritmo se realizó teniendo en cuenta estos tipos de funciones y los parámetros que afectan a la estrategia de búsqueda del mismo. En este segundo análisis se hizo especial hincapié en la poca presión selectiva y el alto nivel de exploración que presenta la configuración original. Modificando la estrategia de búsqueda para tratar de modificar los niveles originales se mejoraron los resultados de todas las funciones analizadas. Sin embargo, algunas de ellas continúan sin alcanzar el valor de CPEM considerado como óptimo. Todas ellas comparten la característica de ser de tipo *No-Separable* y de alta dimensionalidad. Por lo tanto, es posible concluir que este algoritmo puede presentar problemas cuando tratan de resolverse funciones de este tipo.

5.4 Algoritmos macroevolutivos (MA)

5.4.1 Descripción del algoritmo

Los Algoritmos Macroevolutivos (MA) son un nuevo modelo de AE propuesto por Marín y Solé [Marin and Sole, 1998]. En este modelo, los individuos de la población se conocen como especies y el funcionamiento del mismo sigue la dinámica de un ecosistema basándose en las relaciones entre dichas especies. El modelo biológico de la macroevolución simula la extinción y diversificación a gran escala. Las relaciones entre especies son esenciales para determinar el nuevo estado de cada especie en cada generación. El estado de una especie i se define como:

$$S_i(t+1) = \begin{cases} 1 & \text{si el estado es } \textit{vivo} \\ 0 & \text{si el estado es } \textit{extinto} \end{cases} \tag{5.6}$$

Este estado depende de las relaciones entre las especies. Estas relaciones se representan en la matriz de conectividad W, donde cada termino $W_{i,j}(t), i, j = 1, \ldots, NP$ mide la influencia de la especie i en la especie j en cada generación t. Al final de cada generación, todas las especies extinguidas son reemplazadas, o bien por nuevas soluciones, o bien por atraidas hacia especies supervivientes.

Más detalladamente, los pasos que sigue el algoritmo macroevolutivo en cada generación son los siguientes:

1. **Operador de extinción:** permite calcular los individuos supervivientes a través de sus relaciones como una suma de premios y castigos. El estado S_i de una especie viene dado por la fórmula:

$$S_i(t+1) = \begin{cases} 1 & \text{si } \sum_{j=1}^{p} W_{i,j}(t) \geq 0 \\ 0 & \text{en otro caso} \end{cases} \tag{5.7}$$

   donde t es la generación actual y $W_{i.j} = W(p_i, p_j)$ se calcula según la fórmula:

$$W_{i,j} = \frac{f(p_i) - f(p_j)}{|p_i - p_j|} \tag{5.8}$$

2. **Operador de colonización:** una vez determinadas las especies que se extinguen y las que sobreviven, este operador se aplica sobre las extintas para generar nuevas soluciones. Con probabilidad τ, establecida por el usuario, se genera una nueva solución o se

Operador	Parámetro	Valor
Colonización	τ	Lineal
	ρ	0.5

Tabla 5.16: Parámetros de configuración utilizados para la caracterización del MA

explotan las soluciones supervivientes, de modo que se elige uno de los supervivientes para que la especie extinta sea atraída hacia ella. El parámetro τ actúa como "temperatura", puede decrecer a medida que pasan las generaciones imitando un proceso de recocido simulado. Es decir, cuando la temperatura τ es baja, la aleatoriedad es baja y, como consecuencia, hay una tendencia a aumentar la explotación alrededor de los supervivientes y reducir la exploración de nuevas especies. De forma matemática el operador de colonización es el siguiente:

$$p_i(t+1) = \begin{cases} p_r(t) + \rho\lambda(p_r(t) - p_i(t)) & \text{si } \xi > \tau \\ p_n(t) & \text{si } \xi \leq \tau \end{cases} \tag{5.9}$$

donde $\xi \in [0, 1]$ es un número aleatorio y $\lambda \in [-1, 1]$ es también un número aleatorio, ambos con distribución uniforme. ρ y τ son parámetros del algoritmo establecidos por el usuario. τ se explicó con anterioridad y ρ describe el máximo radio de atracción entre especies.

El MA ha sido utilizado para la resolución de problemas de aplicaciones reales, fundamentalmente en la evolución de redes de neuronas artificiales para controladores de robots autónomos [Becerra et al., 2003, Becerra and Reyes, 2005, Becerra et al., 2005a, Becerra et al., 2005b]. El pseudocodigo del MA que se ha sido caracterizado en este trabajo se presenta en el algoritmo 6. Los operadores serán configurados utilizando los valores de los parámetros propuestos por los autores [Marin and Sole, 1998] y que se muestran en la tabla 5.16. De nuevo, los cromosomas se codifican en el rango [-1.0:1.0] y la implementación de este algoritmo está disponible en la librería JEAF (www.gii.udc.es/jeaf).

5.4.2 Caracterización del algoritmo MA

Como se comentó en la introducción del capítulo y de la misma forma que se ha realizado para los algoritmos caracterizados anteriormente, partimos de que las dos primeras etapas del procedimiento de caracterización ya han sido llevadas a cabo y se ha seleccionado el conjunto de funciones prueba caracterizado en la sección 4.1.2 cuyas propiedades se resumen en las tablas 4.3, 4.4 y 4.5. En cuanto a las medidas de error y rendimiento, se utilizarán el CPEM, el SR y el G_{NE}. Describimos a continuación la tercera etapa del procedimiento propuesto, ejecución y análisis de resultados.

Ejecución de las pruebas

Para la ejecución de las pruebas se han seguido las indicaciones descritas en 4.3.1. En la tabla 5.16 se muestran los parámetros escogidos para el MA siguiendo las indicaciones de [Marin and Sole, 1998], de tal forma que el algoritmo se configura para proporcionar el mejor rendimiento promedio en problemas de optimización con parámetros reales. El único parámetro que no prefija el desarrollador es el tamaño de población, que se obtiene tras un

Algoritmo 6 Esquema básico de los algoritmos macroevolutivos.

Entrada:

NP, número de individuos de la población.
D, número de genes de cada cromosoma.
t_{max}, número máximo de generaciones.
ρ, máximo radio de atracción entre especies.
τ, acta como temperatura, determina cuando se explota o se explora.

$t \leftarrow 0$
Generar la población inicial de forma aleatoria: $\mathbf{x}_t^i, \forall i, i = 1, \ldots, NP$
Evaluar $f(\mathbf{x}_t^i), \forall i, i = 1, \ldots, NP$
for $t = 1$ hasta t_{max} **do**
  // Cálculo del estado de cada especie
  for $i = 1$ hasta NP **do**
    for $j = i$ hasta NP **do**
      $W_{i,j}(t) = \frac{f(p_i) - f(p_j)}{|p_i - p_j|}$
      $W_{j,i}(t) = -W_{i,j}(t)$
    end for
    $S_i(t+1) = \begin{cases} 1 & \text{si } \sum_{j=1}^{p} W_{i,j}(t) \geq 0 \\ 0 & \text{en otro caso} \end{cases}$
  end for
  // Operador de colonización
  for $i = 1$ hasta NP **do**
    if $S_i(t+1) = 0$ **then**
      $p_i(t+1) = \begin{cases} p_b(t) + \rho\lambda(p_b(t) - p_i(t)) & \text{si} \xi > \tau \\ p_n(t) & \text{si} \xi \leq \tau \end{cases}$
    end if
  end for
  $t \leftarrow t + 1$
end for

análisis exhaustivo para obtener el óptimo en cada función del conjunto de prueba. En el siguiente apartado se analizan los resultados obtenidos en este caso:

Análisis poblacional La tercera columna de la tabla 5.17 muestra el tamaño de población que obtuvo los mejores resultados para cada dimensión de cada función del conjunto de prueba. Analizaremos estos resultados centrándonos en la modalidad de las funciones. En general, una función multimodal necesita un mayor balance de exploración que una función unimodal, y uno de los parámetros que afecta al nivel de exploración de un algoritmo es el tamaño de población utilizado.

Teniendo en cuenta las funciones unimodales, desde la función *Axis Parallel* hasta la función *Zakharov*, el número de individuos utilizado es, en general, bajo. La mayor parte de las funciones requieren poblaciones de tamaño 10, aparecen algunos casos donde la población es de tamaño 30 o 50 y el resto de los casos, con tamaños de población mayor que 50, pueden ser considerados como excepciones y deben ser analizados en detalle. Estos casos excepcionales incluyen: las funciones *SumOf*, *Colville* y *Perm* en todas las dimensiones analizadas. El nivel medio de *CPEM* para los casos excepcionales en cada uno de los tamaños poblacio-

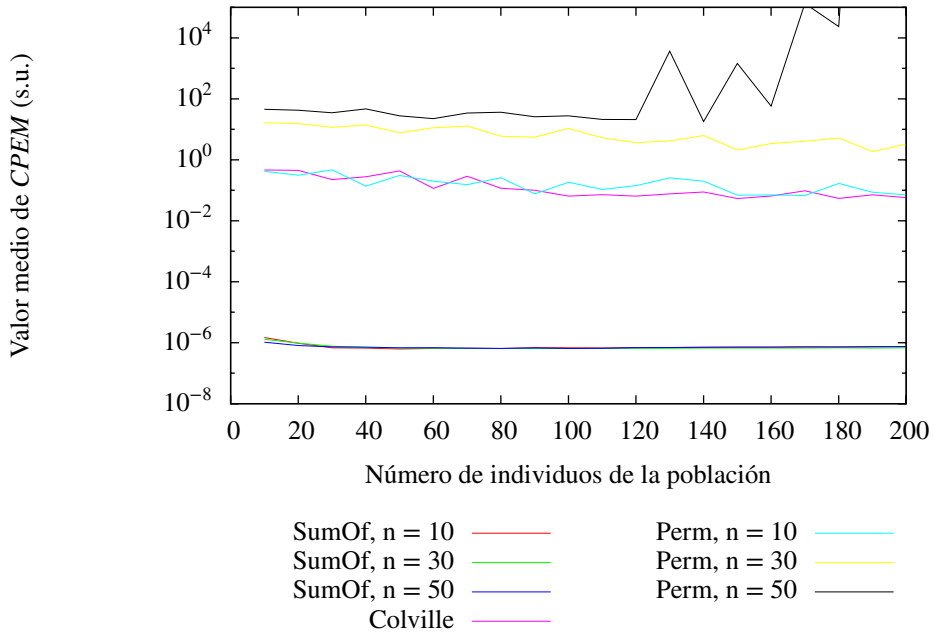


Figura 5.14: Análisis de los casos excepcionales en las funciones unimodales en el análisis poblacional del MA

nales estudiados se muestra en la figura 5.14. Como se puede observar en dicha figura, en el caso de la función *SumOf* para poblaciones de tamaño 10, 20 y 30 el algoritmo no alcanza el valor establecido como óptimo, y sí lo hace a partir de tamaño de población 40. En la función *Colville*, todos los tamaños de población analizados, hasta población 100, obtienen un valor similar de CPEM. Para terminar, en la función *Perm* en dimensiones 10 y 30 el valor de CPEM es similar en todos los tamaños de población analizados y en dimensión 50, hasta 160 se mantiene el mismo valor de CPEM. Los tamaños de población grandes de estas 3 funciones son debidos al criterio utilizado de seleccionar aquel con menor CPEM absoluto, pero en los tres casos se podrían utilizar tamaños por debajo de 50 individuos sin un decremento notable en el valor de CPEM. Por tanto, se puede concluir que en las funciones unimodales, el tamaño de población que utiliza el MA se sitúa entre 10 y 50.

En las funciones multimodales no se observa una tendencia clara. En baja dimensionalidad, el tamaño de población no es determinante a la hora de resolver las funciones. Estas fueron analizadas con tamaños de población entre 10 y 100 individuos y en todas las ejecuciones realizadas el valor de CPEM obtenido no presenta diferencias significativas. Por otro lado, en las funciones de alta dimensionalidad es necesario un análisis más detallado para obtener alguna conclusión con respecto al tamaño de la población. En dimensión 10, todas las funciones analizadas presentan el mismo valor de CPEM con independencia del tamaño de población utilizado. En dimensión 30, los valores de CPEM en las funciones *Rastrigin*, *Schwefel*, *Griewank*, *Penalized 2* y *Rosenbrock* para todos los tamaños de población analizados son similares. En el resto de las funciones, un tamaño de población mayor de 40 individuos es suficiente, excepto en las funciones *Levy* y *Cosine* donde son necesarios más de 100 individuos. En dimensión 50, el número mínimo de individuos se incrementa hasta 50, excepto en los mismo casos que en dimensión 30 (funciones *Levy* y *Cosine*) donde son necesarios más de 200 individuos. A modo de conclusión, se observa que un tamaño de población similar a la dimensión del problema es suficiente en todos los casos, excepto en dos.

Análisis de resultados

Como en los algoritmos anteriores, la caracterización del MA se ha realizado ejecutando el algoritmo sobre las funciones del conjunto de prueba seleccionadas para este trabajo. Para la ejecución de las pruebas se utilizaron las condiciones establecidas en el procedimiento que se explica en la sección 4.3.1. Los resultados obtenidos se muestran en la tabla 5.17.

Tabla 5.17: Resultados obtenidos por el MA en las funciones del conjunto de prueba

Function	D	N	SR	CPEM	GNE
Axis Parallel	10	10	1.00	$7.47e{-7} \pm 1.34e{-8}$	$2.05e{-6} \pm 5.46e{-7}$
	30	10	1.00	$8.28e{-7} \pm 1.38e{-8}$	$9.22e{-7} \pm 2.71e{-7}$
	50	10	1.00	$8.72e{-7} \pm 1.97e{-8}$	$4.48e{-7} \pm 7.89e{-8}$
Schwefel 2.22	10	10	1.00	$8.35e{-7} \pm 8.85e{-9}$	$1.70e{-8} \pm 6.22e{-9}$
	30	10	1.00	$8.79e{-7} \pm 7.02e{-9}$	$6.52e{-9} \pm 2.36e{-9}$
	50	20	1.00	$9.17e{-7} \pm 6.31e{-9}$	$4.23e{-9} \pm 1.37e{-9}$
Sphere	10	10	1.00	$7.26e{-7} \pm 1.29e{-8}$	$3.32e{-6} \pm 2.46e{-7}$
	30	10	1.00	$7.73e{-7} \pm 6.43e{-9}$	$1.86e{-6} \pm 4.53e{-8}$
	50	10	1.00	$7.92e{-7} \pm 5.37e{-9}$	$1.42e{-6} \pm 1.53e{-8}$
Step	10	10	1.00	$4.18e{-7} \pm 4.06e{-8}$	$3.84e{-3} \pm 8.93e{-4}$
	30	30	1.00	$6.02e{-7} \pm 4.84e{-8}$	$3.13e{-3} \pm 2.82e{-4}$
	50	30	1.00	$7.00e{-7} \pm 7.40e{-8}$	$3.09e{-3} \pm 2.16e{-4}$
SumOf	10	50	1.00	$6.23e{-7} \pm 3.83e{-8}$	$6.44e{-2} \pm 1.71e{-2}$
	30	90	1.00	$6.42e{-7} \pm 4.04e{-8}$	$1.96e{-1} \pm 2.54e{-2}$
	50	70	1.00	$6.58e{-7} \pm 3.54e{-8}$	$2.91e{-1} \pm 1.82e{-2}$
Easom	2	10	1.00	$5.52e{-7} \pm 4.19e{-8}$	$8.33e{-6} \pm 3.78e{-6}$
Schwefel 2.21	10	10	0.04	$1.17e{-6} \pm 7.18e{-7}$	$6.16e{-6} \pm 1.45e{-6}$
	30	10	0.00	$1.67e{-3} \pm 1.11e{-3}$	$1.13e{-5} \pm 7.62e{-6}$
	50	10	0.00	$4.50e{-1} \pm 5.87e{-1}$	$3.67e{-3} \pm 5.17e{-3}$
Colville	4	1	0.00	$6.41e{-2} \pm 7.21e{-2}$	$1.99e{-2} \pm 1.68e{-2}$
Matyas	2	10	1.00	$4.29e{-7} \pm 4.32e{-8}$	$5.59e{-4} \pm 2.42e{-4}$
Perm	10	170	0.00	$6.74e{-2} \pm 9.08e{-2}$	$1.96e{-2} \pm 8.41e{-3}$
	30	150	0.00	$2.10e{+0} \pm 3.36e{+0}$	$5.59e{-3} \pm 1.82e{-3}$
	50	140	0.00	$1.79e{+1} \pm 2.91e{+1}$	$1.75e{-2} \pm 4.71e{-3}$
Schwefel 1.2	10	10	0.84	$1.17e{-6} \pm 7.18e{-7}$	$6.16e{-6} \pm 1.45e{-6}$
	30	10	0.00	$6.74e{-2} \pm 5.05e{-2}$	$8.62e{-4} \pm 2.91e{-4}$
	50	10	0.00	$4.35e{+0} \pm 1.50e{+0}$	$5.52e{-3} \pm 1.07e{-3}$
Zakharov	10	10	1.00	$7.59e{-7} \pm 2.04e{-8}$	$4.25e{-5} \pm 1.47e{-6}$
	30	10	0.36	$1.72e{-6} \pm 1.28e{-6}$	$3.07e{-5} \pm 9.55e{-6}$
	50	10	0.00	$1.50e{-4} \pm 6.46e{-5}$	$2.25e{-4} \pm 5.18e{-5}$
Aluffi-Pentini's	2	10	1.00	$4.42e{-7} \pm 4.10e{-8}$	$1.48e{-4} \pm 1.27e{-4}$
Becker and Lago	2	20	1.00	$5.80e{-7} \pm 4.57e{-8}$	$6.13e{-1} \pm 3.30e{-1}$
Bohachevsky 1	2	10	1.00	$5.86e{-7} \pm 4.53e{-8}$	$5.34e{-6} \pm 3.87e{-6}$
Cosine Mixture	10	10	1.00	$6.15e{-7} \pm 4.86e{-8}$	$9.55e{-5} \pm 1.21e{-5}$
	30	160	1.00	$8.97e{-7} \pm 4.53e{-9}$	$5.12e{-5} \pm 2.09e{-6}$
	50	160	0.92	$1.78e{-2} \pm 6.38e{-2}$	$2.64e{-2} \pm 3.12e{-2}$
Rastrigin	10	180	0.04	$2.24e{+0} \pm 1.23e{+0}$	$8.46e{-2} \pm 3.68e{-2}$
	30	150	0.00	$2.60e{+1} \pm 7.37e{+0}$	$1.80e{-1} \pm 2.64e{-2}$
	50	220	0.00	$5.63e{+1} \pm 1.17e{+1}$	$3.10e{-1} \pm 2.93e{-2}$
Schwefel	10	190	0.00	$2.64e{+2} \pm 1.33e{+2}$	$6.18e{-1} \pm 2.29e{-1}$
	30	170	0.00	$2.60e{+3} \pm 4.25e{+2}$	$9.07e{-1} \pm 1.13e{-1}$
	50	270	0.00	$4.92e{+3} \pm 6.26e{+2}$	$9.23e{-1} \pm 7.51e{-2}$
Ackleys	10	10	1.00	$8.51e{-7} \pm 9.22e{-9}$	$8.81e{-9} \pm 2.42e{-9}$
	30	30	1.00	$9.32e{-7} \pm 3.79e{-9}$	$7.90e{-9} \pm 1.07e{-10}$
	50	70	0.88	$9.99e{-7} \pm 2.34e{-8}$	$7.89e{-9} \pm 1.72e{-10}$
Griewank	30	90	0.76	$3.45e{-3} \pm 6.64e{-3}$	$7.69e{-4} \pm 1.08e{-3}$

Tabla 5.17 -- continua desde la página anterior

Function	D	N	SR	CPEM	GNE
	50	120	0.80	$2.17e{-3} \pm 4.74e{-3}$	$3.06e{-4} \pm 6.35e{-4}$
Levy	10	10	1.00	$5.90e{-7} \pm 1.85e{-8}$	$1.30e{-4} \pm 1.05e{-5}$
	30	140	1.00	$8.38e{-7} \pm 1.71e{-7}$	$2.96e{-3} \pm 1.44e{-2}$
	50	290	0.96	$3.96e{-2} \pm 1.94e{-1}$	$6.77e{-3} \pm 1.86e{-2}$
Penalized 1	10	10	1.00	$6.59e{-7} \pm 2.95e{-8}$	$4.59e{-5} \pm 4.11e{-6}$
	30	40	1.00	$8.05e{-7} \pm 2.98e{-8}$	$4.54e{-5} \pm 8.46e{-7}$
	50	80	0.84	$2.73e{-2} \pm 7.88e{-2}$	$3.74e{-3} \pm 7.30e{-3}$
Penalized 2	10	10	1.00	$6.83e{-7} \pm 1.50e{-8}$	$2.08e{-5} \pm 4.16e{-6}$
	30	10	1.00	$7.52e{-7} \pm 4.27e{-8}$	$1.17e{-5} \pm 3.26e{-7}$
	50	10	1.00	$7.69e{-7} \pm 1.97e{-8}$	$4.60e{-5} \pm 1.85e{-4}$
Beale	2	10	1.00	$5.00e{-7} \pm 6.30e{-8}$	$5.24e{-4} \pm 4.23e{-4}$
Bohachevsky 2	2	10	1.00	$5.92e{-7} \pm 4.09e{-8}$	$5.08e{-6} \pm 3.65e{-6}$
Dekkers and Aarts	2	10	1.00	$3.62e{-7} \pm 5.20e{-8}$	$5.77e{-1} \pm 5.20e{-1}$
Goldstein Price	2	10	1.00	$5.24e{-7} \pm 4.16e{-8}$	$2.73e{-5} \pm 1.04e{-5}$
Griewank	10	90	0.20	$2.50e{-2} \pm 1.76e{-2}$	$4.59e{-3} \pm 2.63e{-3}$
Hartman 3	3	10	1.00	$2.39e{-7} \pm 3.00e{-8}$	$6.32e{-3} \pm 2.97e{-3}$
Hartman 6	6	40	0.84	$2.03e{-2} \pm 4.66e{-2}$	$1.51e{-1} \pm 3.34e{-1}$
Rosenbrock	10	50	0.00	$6.15e{+0} \pm 1.07e{+0}$	$2.87e{-2} \pm 1.39e{-3}$
	30	140	0.00	$3.12e{+1} \pm 1.64e{+1}$	$3.47e{-2} \pm 1.57e{-2}$
	50	10	0.00	$5.55e{+1} \pm 2.15e{+1}$	$3.35e{-2} \pm 7.14e{-3}$
Kowaliks	4	90	0.00	$3.68e{-4} \pm 1.11e{-4}$	$3.08e{-1} \pm 1.39e{-1}$
Shekel Family 5	4	30	1.00	$3.87e{-7} \pm 2.89e{-8}$	$2.96e{-5} \pm 9.20e{-6}$
Shekel Family 7	4	10	1.00	$5.00e{-7} \pm 3.07e{-8}$	$2.12e{-4} \pm 1.35e{-4}$
Shekel Family 10	4	20	1.00	$5.56e{-7} \pm 3.18e{-8}$	$1.99e{-4} \pm 7.72e{-5}$
Shekels Foxholes	2	10	1.00	$3.16e{-7} \pm 5.68e{-8}$	$1.98e{-3} \pm 8.47e{-4}$
SixHump Camel Back	2	10	1.00	$4.69e{-7} \pm 4.81e{-8}$	$1.14e{-1} \pm 1.03e{-1}$

Configuración básica Los resultados obtenidos por el MA sobre el conjunto de funciones de prueba utilizado en este trabajo se muestran en la tabla 5.17. Analizando estos resultados en términos de separabilidad no es posible obtener una conclusión clara acerca de su comportamiento. En el conjunto de funciones prueba existen once funciones de tipo *L-Separable* y el MA resuelve ocho de ellas con un SR del 100% en todas las dimensiones analizadas. En la función *Cosine* falla dos ejecuciones (SR 92%) cuando se ejecuta con dimensión 50 y en las funciones *Rastrigin* y *Schwefel* no resuelve ninguna de las ejecuciones con independencia de la dimensión. Analizando con mayor detalle este tipo de funciones L-separables, se observa que las que resuelve siempre son de tipo unimodal o multimodal de baja dimensionalidad, mientras que en las presenta problemas son todas multimodales de alta dimensionalidad. Más adelante analizaremos estas funciones desde el punto de vista de la modalidad para determinar cual es el problema concreto del MA.

El siguiente tipo de funciones a analizar en términos de separabilidad son las *NL-Separables*. Existen siete funciones de este tipo en el conjunto de prueba. De ellas, el MA resuelve todas las ejecuciones de dos, la función *Easom* y la función *Penalized 2* y falla en algunas ejecuciones de cuatro, la función *Ackleys* en dimensión 50 (SR 88%), la función *Griewank* en dimensión 30 (SR 76%) y dimensión 50 (SR 80%), la función *Levy* en dimensión 50 (SR 96%) y la función *Penalized 1* en dimensión 50 (SR 84%). Por último, no resuelve ninguna ejecución de la función *Schwefel 2.21* en ninguna de las dimensiones analizadas.

El resto de funciones, concretamente diecinueve, son de tipo *No-Separable*. El MA resuelve once de ellas con un SR del 100%, siendo todas de baja dimensionalidad, tanto unimo-

dales como multimodales. Resuelve algunas ejecuciones de cuatro de ellas, concretamente las funciones *Schwefel 1.2* (SR 84%, 0% y 0%, en 10, 30, y 50 dimensiones, respectivamente), *Zakharov* (SR 100%, 36% y 0%, en 10, 30, y 50 dimensiones, respectivamente), *Griewank* (SR 20% en 10 dimensiones) y *Hartman 6* (SR 84%). Por último, no resuelve ninguna ejecución de las funciones *Colville*, *Perm*, *Rosenbrock* y *Kowaliks*.

Como se mencionó al inicio de este análisis, tratando la propiedad de separabilidad de forma independiente, no existe una tendencia clara en la respuesta del algoritmo MA. La estrategia de búsqueda que utiliza este algoritmo no explota las direcciones ortogonales ni tampoco modifica los genes de los cromosomas teniendo en cuenta las dependencias entre ellos. En concreto, si una especie no sobrevive se genera una nueva. Esto puede realizarse de dos maneras, o bien de forma totalmente aleatoria o bien a partir de un superviviente. Como consecuencia, el movimiento en el espacio de calidad se realiza en todas las dimensiones al mismo tiempo y, por tanto, el número de genes que sobrevive entre generaciones no puede ser controlado. Para obtener análisis más preciso del comportamiento del MA, es necesario analizar en detalle las funciones en términos de modalidad.

El MA no resuelve nunca únicamente dos funciones de tipo *L-Separables*, la función *Rastrigin* y la función *Schwefel*, ambas multimodales. En ambos casos, tanto el valor de CPEM como el valor de G_{NE} son elevados. De hecho, si se comparan los valores de G_{NE} con la distancia máxima entre centros de atracción obtenida tras analizar las funciones con el algoritmo presentado en el capítulo 4 (ver tabla 4.4), las soluciones obtenidas por el algoritmo distan mucho del óptimo global y se acercan a los centros de atracción más alejados. Las diferencias entre estas funciones y la función *Cosine*, que sí resuelve el MAcon un SR del 92%, son dos. Por un lado, el número de centros de atracción. La función *Cosine* presenta menos centros de atracción que las funciones *Rastrigin* y *Schwefel*. Por otro lado, el tamaño del centro de atracción óptimo, que en el caso de las funciones no resueltas es considerablemente pequeño.

En las funciones de tipo *NL-Separables*, el MA presenta mayores problemas en las unimodales que en las multimodales. En este último tipo de funciones resuelve casi todas las ejecuciones realizadas, siempre con un SR mayor que el 75%. Cabe destacar el caso de la función *Griewank*, donde se obtienen los peores resultados de este tipo de funciones. En esta función multimodal, al igual que ocurría con las funciones *Rastrigin* y *Schwefel*, el centro de atracción óptimo tiene un tamaño considerablemente pequeño, concretamente de $2.00e{-}4$ unidades. Por el contrario, y comparando de nuevo con estas dos funciones, el número de centros de atracción no es elevado y, de hecho, esta función, junto con la función *Rosenbrock*, es la que presenta un menor número de centros de atracción. Como consecuencia, es posible concluir que, dentro de las funciones multimodales, la característica que afecta en mayor medida al MA es el tamaño del centro de atracción óptimo. Para terminar con el análisis en modalidad de las funciones *NL-Separables*, la función *Schwefel 2.21* es unimodal con un camino al óptimo muy largo, concretamente es la segunda función con el camino más largo de las funciones unimodales analizadas (ver 4.5). Este algoritmo no resuelve ninguna ejecución en ninguna de las dimensiones analizadas. En dimensión 10, el valor de CPEM obtenido, $1.17e{-}6$, se sitúa muy cerca del valor establecido para considerar una función resuelta, 10^{-6}, lo cual permite anticipar que suavizando el criterio de parada este algoritmo mejoraría sus resultados en cuanto a SR. Cuando se utilizan 30 dimensiones, el valor de CPEM es mayor, $1.67e{-}3$, sin embargo, el valor de G_{NE} es solamente un orden de magnitud mayor que para dimensión 30. Los valores obtenidos para dimensión 50 en cuanto a G_{NE} son algo mayores. Esta función será analizada detalladamente en el siguiente apartado de esta sección para, al igual que se hizo con el RCGAen las funciones unimodales, determinar si el problema que tiene este algoritmo para no resolver la función es el criterio de parada utilizado.

Como se comentó anteriormente, en las funciones *No-Separables* es donde el algoritmo tiene mayores problemas. El MA es capaz de resolver funciones *No-Separables* de baja dimensionalidad cuyas características en cuanto a modalidad no presenten dificultades, como ocurre con la función *Matyas*, que es unimodal con un camino al óptimo corto. Así, del conjunto de funciones de baja dimensionalidad, el MA no resuelve tres: la función *Colville*, la función *Kowaliks* (ambas de dimensión 4) y la función *Hartman 6*, de dimensión 6. La función *Colville* es unimodal y presenta el tercer camino al óptimo de mayor longitud y, como se ha visto anteriormente, una longitud al óptimo elevada degrada el rendimiento del MA. Las dos funciones restantes son multimodales. En ambos casos, el centro de atracción óptimo no es el de mayor tamaño. De las cinco funciones *No-Separables* escalables, la función *Perm*, la función *Schwefel 1.2*, la función *Zakharov*, la función *Griewank* D10 y la función *Rosenbrock*, el MA resuelve algunas ejecuciones de las funciones *Schwefel 1.2*, *Zakharov* y *Griewank*, siempre en dimensión 10. Si se incrementa el número de dimensiones el algoritmo no resuelve ninguna ejecución de las realizadas. Las tres primeras funciones son unimodales. De ellas, la que mejores resultados obtiene es la función *Zakharov*, que es la que tiene el camino al óptimo de menor longitud. Estas tres funciones serán analizadas posteriormente para comprobar si suavizando el criterio de parada podrían resolverse.

Para terminar, las funciones *Griewank* y *Rosenbrock* son multimodales. La modalidad de la función *Griewank* ya ha sido analizada. En este caso, el tamaño del centro de atracción óptimo unido a la no separabilidad son las características que provocan el rendimiento pobre del MA. La función *Rosenbrock* no presenta ninguna característica en cuanto a modalidad que afecte al rendimiento del MA. Tras un análisis detallado, únicamente se encontraron nueve centros de atracción. De estos nueve centros de atrcción, dos de ellos ocupan casi la totalidad del espacio de calidad, el tamaño del centro de atracción óptimo es considerablemente alto ($3.76e{-}1$ unidades) y la distancia entre los dos centros de atracción principales no es destacablemente alta. Por tanto, es posible concluir que la no separabilidad es la característica que provoca el mal comportamiento del MA en esta función.

Resumiendo, tras el análisis de los resultados obtenidos por el MA en el conjunto de funciones de prueba utilizado en este trabajo es posible realizar la siguiente caracterización del algoritmo:

- Su rendimiento se degrada a medida que aumenta la dimensionalidad del problema sin importar el tipo de separabilidad. Esto ocurre en mayor medida en las funciones de tipo *No-Separables*.

- En términos de modalidad, este algoritmo no es recomendable en funciones unimodales cuando la longitud del camino al óptimo es mayor de 200 pasos.

- En funciones multimodales se comporta de manera satisfactoria si no existe mucha distancia entre los centros de atracción de la función o si el centro de atracción óptimo es amplio comparando con el resto de centros de atracción. Cuando estos centros de atracción se encuentran espaciados sobre el espacio de calidad o el centro de atracción óptimo no es el de mayor tamaño, el MA no es recomendable.

Variaciones sobre la configuración básica Una vez que el algoritmo ha sido analizado utilizando su configuración básica y han sido detectados los tipos de funciones en los cuales presenta mayores problemas, en esta sección trataremos de modificar el comportamiento del algoritmo ajustando sus parámetros tratando de resolver estos casos problemáticos. Como en

los tres algoritmos previamente analizados, este ajuste de parámetros se realizará teniendo en cuenta las cuatro propiedades principales que definen la estrategia de búsqueda de un AE: balance exploración / explotación, presión selectiva, tamaño del paso de mutación y direcciones de búsqueda.

Las funciones no resueltas por este algoritmo son las siguientes:

- Unimodales: el MA no resuelve las funciones *Schwefel 2.21*, *Colville*, *Perm*, *Schwefel 1.2* y*Zakharov*, todas ellas de camino largo. En cuanto a separabilidad, la función *Schwefel 2.21* es de tipo *NL-Separable* y el resto *No-Separables*.

- Multimodales: las funciones *Rastrigin* y *Schwefel*, ambas *L-Separables*, y las funciones *Griewank*, *Rosenbrock* y *Kowaliks* todas ellas de tipo *No-Separables*.

Para comprobar qué ocurre en las ejecuciones no resueltas se analizarán, en primer lugar, las gráficas de evolución que se muestran en las figuras 5.15 y 5.16. La primera corresponde a las funciones *Schwefel 2.21*, *Schwefel 1.2* y *Zakharov*. Como se puede ver en dichas gráficas, las ejecuciones realizadas no alcanzan el valor establecido como óptimo (10^{-6}) en el número de FEs establecido y en ningún caso presenta convergencia prematura. Por tanto, se puede suponer que aumentando la velocidad de convergencia el MA sería capaz de resolverlas.

En la figura 5.16 se muestran las gráficas de evolución de las funciones *Perm* y *Colville*. Como se observa en dichas gráficas, el comportamiento del algoritmo es diferente. Al igual que ocurría con las tres funciones anteriores, existen ejecuciones en las cuales la calidad del mejor individuo no ha convergido hacia un valor en el número de FEs establecido como límite. Sin embargo, existen algunas ejecuciones en las que el valor de calidad del mejor individuo deja de mejorar antes de alcanzar el límite de FEs.

Este algoritmo presenta una velocidad de convergencia lenta. Para tratar de incrementarla existen diferentes modificaciones que pueden realizarse sobre la configuración original. Estas modificaciones son:

- Aumentar la presión selectiva modificando el operador de colonización, utilizando como especie hacia la que atraer las soluciones extintas la mejor especie de la población y no una aleatoria [Marín, 2005]. A partir de aquí esta modificación se denominará estrategia *best*.

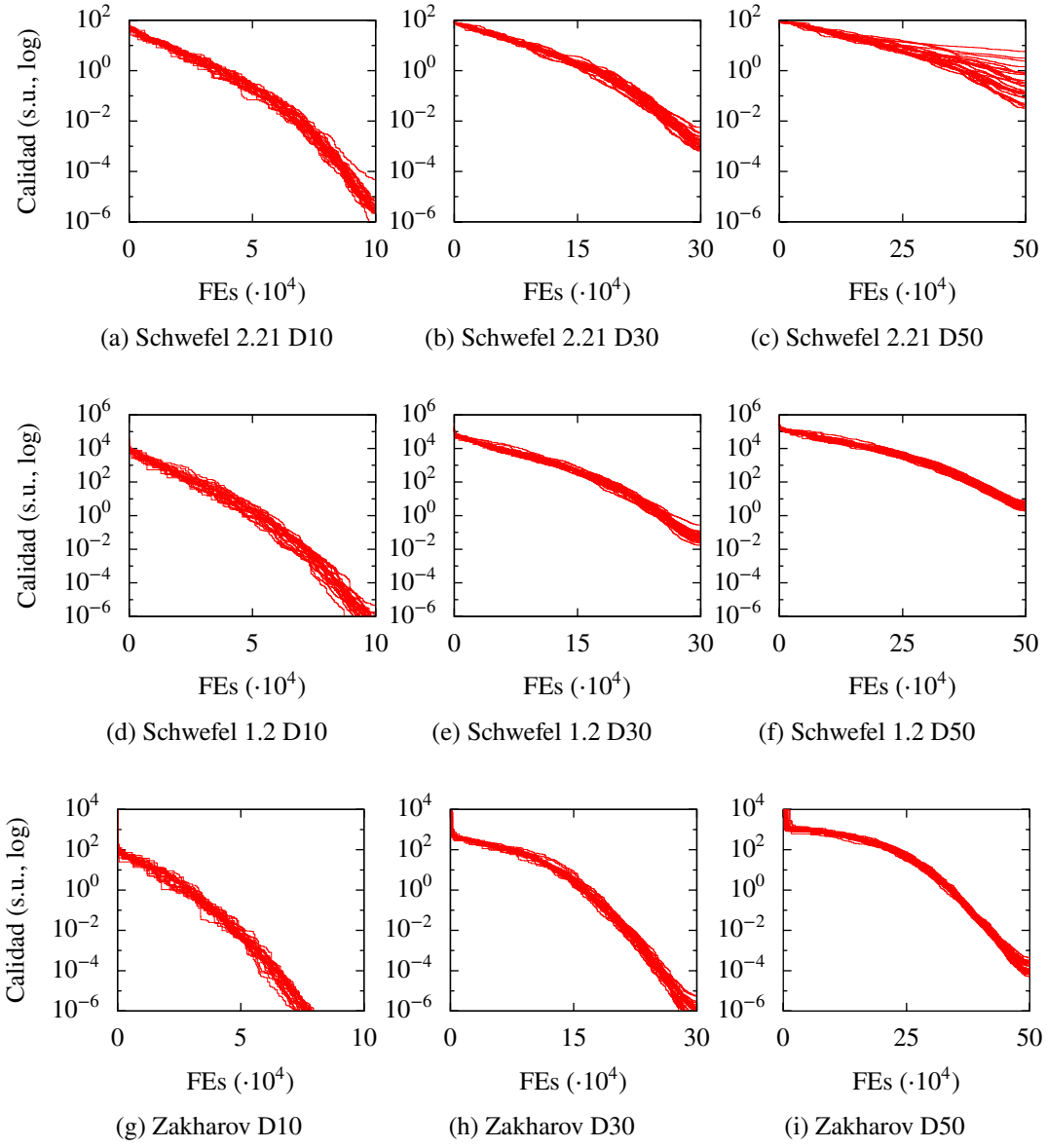
- Aumentar el nivel de explotación del algoritmo. Esto puede llevarse a cabo de dos formas:

  - Modificando la función que genera el valor de τ.

$$\tau = 1.0 - \frac{t}{0.7 \cdot t_{max}} \tag{5.10}$$

  - Disminuyendo el valor de ρ, de esta forma los cambios se concentran alrededor de las especies supervivientes.

Todas las propuestas anteriores han sido ejecutadas sobre las funciones unimodales no resueltas. Por cuestión de espacio sólo se comentarán aquellos casos que hayan supuesto una mejora en los resultados originales. Concretamente, para las funciones con un camino al óptimo largo (la función *Perm* y la función *Schwefel 2.21*) aumentar el nivel de explotación, modificando la función que genera τ acelera la velocidad de convergencia del algoritmo obteniendo una mejora sobre los resultados originales, aunque únicamente cuando la dimensión



(a) Schwefel 2.21 D10 (b) Schwefel 2.21 D30 (c) Schwefel 2.21 D50

(d) Schwefel 1.2 D10 (e) Schwefel 1.2 D30 (f) Schwefel 1.2 D50

(g) Zakharov D10 (h) Zakharov D30 (i) Zakharov D50

Figura 5.15: Ejecuciones realizadas para las funciones unimodales *Schwefel 2.21*, *Schwefel 1.2* y *Zakharov*. El eje x representa el número de FEs ejecutadas. El eje y representa el valor de calidad del mejor individuo de la población en escala logarítmica.

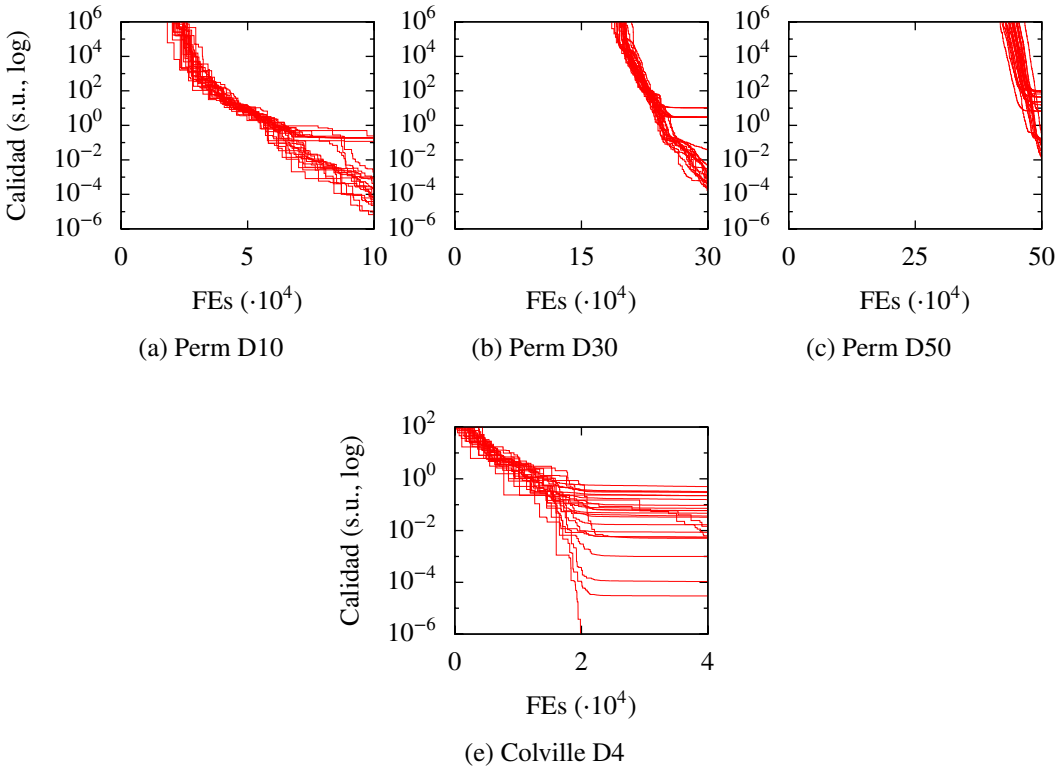


Figura 5.16: Ejecuciones realizadas para las funciones unimodales *Perm* y *Colville*. El eje x representa el número de FEs ejecutadas. El eje y representa el valor de calidad del mejor individuo de la población en escala logarítmica.

Función	D	N	Colonización	SR	CPEM	G_{NE}
Perm	10	170	random	0.00	$6.74e{-2} \pm 9.08e{-2}$	$1.96e{-2} \pm 8.41e{-3}$
			$\tau = 0.7$	0.08	$3.08e{-2} \pm 7.00e{-2}$	$1.68e{-2} \pm 6.91e{-3}$
Schwefel 2.21	10	10	random	0.76	$1.24e{-6} \pm 8.08e{-7}$	$8.47e{-9} \pm 5.99e{-9}$
			$\tau = 0.7$	0.92	$9.60e{-7} \pm 1.16e{-7}$	$6.76e{-9} \pm 1.14e{-9}$

Tabla 5.18: Comparación de los resultados del MA en las funciones *Perm* y *Schwefel 2.21* cuando se utiliza la función de τ modificada según la ecuación 5.10.

del problema es 10. En la tabla 5.18 se comparan los resultados obtenidos por el algoritmo MA utilizando la configuración original y aquellos obtenidos con la función τ modificada.

En las funciones que presentan un camino al óptimo de longitud menor (la función *Schwefel 1.2*, la función *Zakharov* y la función *Colville*), la aproximación que mejores resultados proporciona es aquella que disminuye el valor de ρ, de nuevo para aumentar el nivel de explotación de las especies supervivientes. Los resultados de estas tres funciones se muestran en la tabla 5.19, donde se comparan los obtenidos por la configuración original con los que se obtienen utilizando un valor de ρ de 0.25 en lugar del 0.5 original.

A pesar de las modificaciones realizadas el algoritmo MA continua sin resolver todas las funciones unimodales. Del mismo modo que se llevó a cabo con el algoritmo RCGA, este algoritmo se ejecutó incrementando el número de FEs ejecutadas. Realizando esta modificación sobre las condiciones de ejecución el algoritmo es capaz de resolver la función *NL-Separable Schwefel 2.21* y las dos funciones *No-Separables* con camino al óptimo más corto. Los resultados de las funciones *Perm* y *Colville* mejoran, pero el SR continua siendo

Función	D	N	Colonización	SR	CPEM	G_{NE}
Schwefel 1.2	10	10	random	0.84	$1.17e{-6} \pm 7.18e{-7}$	$6.16e{-6} \pm 1.45e{-6}$
			$\rho = 0.25$	1.00	$8.67e{-7} \pm 2.36e{-8}$	$5.60e{-6} \pm 5.46e{-7}$
	30	10	random	0.00	$6.42e{-2} \pm 5.05e{-2}$	$8.62e{-4} \pm 2.91e{-4}$
			$\rho = 0.25$	0.00	$1.05e{-2} \pm 7.88e{-3}$	$3.33e{-4} \pm 1.25e{-4}$
	50	10	random	0.00	$4.35e{+0} \pm 1.50e{+0}$	$5.52e{-3} \pm 1.07e{-3}$
			$\rho = 0.25$	0.00	$1.20e{+0} \pm 2.87e{-1}$	$2.96e{-3} \pm 3.51e{-4}$
Zakharov	10	10	random	1.00	$7.59e{-7} \pm 2.04e{-8}$	$4.25e{-5} \pm 1.74e{-7}$
			$\rho = 0.25$	1.00	$6.53e{-7} \pm 3.05e{-8}$	$3.85e{-5} \pm 3.54e{-6}$
	30	10	random	0.36	$1.72e{-6} \pm 1.28e{-6}$	$3.05e{-5} \pm 6.55e{-7}$
			$\rho = 0.25$	1.00	$8.68e{-7} \pm 1.71e{-8}$	$2.40e{-5} \pm 6.22e{-5}$
	50	10	random	0.00	$1.50e{-4} \pm 6.46e{-5}$	$2.24e{-5} \pm 5.18e{-5}$
			$\rho = 0.25$	0.00	$1.11e{-5} \pm 6.23e{-6}$	$6.07e{-5} \pm 1.67e{-5}$
Colville	4	90	random	0.00	$6.41e{-2} \pm 7.21e{-2}$	$1.99e{-2} \pm 1.68e{-2}$
			$\rho = 0.25$	0.12	$6.05e{-2} \pm 9.28e{-2}$	$1.36e{-2} \pm 1.49e{-2}$

Tabla 5.19: Comparación de los resultados del MA en las funciones *Schwefel 1.2*, *Zakharov* y *Colville* cuando se modifica el valor original de ρ de 0.5 a 0.25

Función	D	N	FEs	SR	CPEM	G_{NE}
Schwefel 2.21	10	10	$100000 \cdot n$	1.00	$7.55e{-7} \pm 1.04e{-8}$	$6.16e{-9} \pm 7.30e{-10}$
	30	10	$100000 \cdot n$	1.00	$8.39e{-7} \pm 7.94e{-9}$	$6.34e{-9} \pm 4.54e{-10}$
	50	10	$100000 \cdot n$	1.00	$8.79e{-7} \pm 4.06e{-9}$	$6.34e{-9} \pm 3.90e{-10}$
Schwefel 1.2	10	10	$100000 \cdot n$	1.00	$6.72e{-7} \pm 1.68e{-8}$	$5.33e{-6} \pm 5.58e{-7}$
	30	10	$100000 \cdot n$	1.00	$8.39e{-7} \pm 8.34e{-9}$	$3.42e{-6} \pm 8.77e{-8}$
	50	10	$100000 \cdot n$	1.00	$9.27e{-7} \pm 7.54e{-9}$	$2.73e{-6} \pm 3.93e{-8}$
Zakharov	10	10	$100000 \cdot n$	1.00	$5.37e{-7} \pm 1.35e{-8}$	$3.93e{-5} \pm 3.16e{-6}$
	30	10	$100000 \cdot n$	1.00	$6.78e{-7} \pm 8.22e{-9}$	$2.37e{-5} \pm 5.16e{-7}$
	50	10	$100000 \cdot n$	1.00	$7.39e{-7} \pm 5.87e{-9}$	$1.87e{-5} \pm 2.11e{-7}$
Perm	10	170	$100000 \cdot n$	0.12	$4.60e{-2} \pm 8.15e{-2}$	$1.76e{-2} \pm 7.89e{-3}$
	30	150	$100000 \cdot n$	0.00	$2.74e{+0} \pm 3.63e{+0}$	$5.88e{-3} \pm 2.43e{-3}$
	50	140	$100000 \cdot n$	0.00	$9.25e{+0} \pm 1.28e{+1}$	$2.98e{-3} \pm 1.11e{-3}$
Colville	4	90	$100000 \cdot n$	0.04	$6.79e{-4} \pm 9.72e{-4}$	$1.63e{-3} \pm 1.45e{-3}$

Tabla 5.20: Resultados obtenidos por el algoritmo MA en las funciones unimodales consideradas en este apartado cuando el número de FEs permitidas se incrementa de $10000 \cdot n$ a $100000 \cdot n$.

inferior al 100%. En la tabla 5.20 se muestran los resultados de estas ejecuciones.

En el caso de las funciones multimodales, el algoritmo ha convergido hacia óptimos locales. Todas las funciones multimodales no resueltas presentan centros de atracción óptimos de tamaño pequeño comparándolos con los demás centros de atracción de su espacio de calidad. Siguiendo la filosofía del procedimiento de caracterización desarrollado en este trabajo, las posibles modificaciones a realizar en este tipo de funciones son las siguientes:

- Aumentar el nivel de exploración, tratando que el algoritmo no converja a un óptimo local. Esto se consigue modificando la función que genera el valor de τ según la ecuación:

$$\tau = 1.0 - \frac{0.8 \cdot t}{t_{max}} \tag{5.11}$$

- Aumentar la presión selectiva para que el algoritmo tienda hacia la mejor solución utilizando la estrategia *best*.

- Disminuir el paso de mutación para aumentar con ello el nivel de exploración. Esto se consigue aumentando el valor de ρ.

Todas las variaciones anteriores fueron probadas en el subconjunto de funciones multimodales no resueltas por el MA. En ninguno de los casos se obtuvo una mejoría clara en los valores de CPEM o SR.

En esta sección, el procedimiento de caracterización desarrollado se ha aplicado a un algoritmo menos conocido y utilizado que los tres anteriores, el MA. En un primer análisis, se utilizaron los parámetros de configuración recomendados en [Marin and Sole, 1998]. A partir de los resultados obtenidos tras este primer análisis fue posible concluir que el algoritmo no se recomienda para funciones de tipo *L-Separables* multimodales de alta dimensionalidad o *No-Separables*, ya sean unimodales o multimodales, de alta dimensionalidad. Analizando estas funciones se procedió a un segundo análisis tratando de mejorar los resultados obtenidos modificando la configuración original del algoritmo. En las funciones unimodales se observó que el problema del algoritmo es la lenta velocidad de convergencia que presenta. Aumentando el nivel de explotación se consiguió incrementar esta velocidad. Sin embargo, el algoritmo continua sin resolver completamente este tipo de funciones. Modificando los criterios de ejecución, permitiendo más evaluaciones de la función de calidad en el criterio de parada el algoritmo resuelve todas las funciones unimodales excepto aquellas que presentan el camino al óptimo más largo. En el caso de las funciones multimodales, ya sean *L-Separables* o *No-Separables*, tras este segundo análisis no se obtuvo ninguna mejora. Por lo tanto, es posible concluir que su velocidad de convergencia es lenta en funciones unimodales y presenta problemas en funciones multimodales de alta dimensionalidad cuando el centro de atracción óptimo tiene un tamaño pequeño.

5.5 Comparación de algoritmos

El último paso de la metodología de análisis de resultados se centra en la comparación del algoritmo estudiado con otros que hayan sido caracterizados previamente. En este caso, la comparación de los cuatro AEs se realizará de manera conjunta al carecer de resultados previos.

La comparación de los AEs se realizará utilizando las tablas de resultados presentadas en cada apartado de análisis correspondiente: la tabla 5.2 para el RCGA, la tabla 5.8 para el CMA-ES la tabla 5.13 para el DE y la tabla 5.17 para el MA. En primer lugar, los resultados del análisis poblacional muestran que el tamaño de la población que cada algoritmo necesita para obtener los mejores valores de CPEM es diferente. En particular, el RCGA necesita más individuos que los otros tres algoritmos, lo cual representa un inconveniente. Debido al test de parada utilizado, basado en el número de llamadas a la función de evaluación, los algoritmos que utilizan menos individuos para sus evoluciones se benefician de una mayor número de generaciones en los procesos de optimización. Como ya se mencionó en la sección 5.1, algunos de los casos considerados como no resueltos por este algoritmo se deben a que el criterio de parada establecido en este procedimiento no permite terminar la evolución. En otras palabras, la población del algoritmo no converge hacia una solución en el tiempo de ejecución permitido. Esto implica que, en un caso práctico, si un usuario quiere utilizar el RCGA deberá permitir al algoritmo ejecutar más evaluaciones que en el caso de utilizar uno de los otros algoritmos.

Teniendo en cuenta el rendimiento de los cuatro algoritmos y los grupos de funciones de prueba utilizados en esta tesis, se puede concluir que la mejor elección es el DE, a menos que la función sea de tipo no separable de alta dimensionalidad o no separable y, al mismo tiempo, multimodal con pocos centros de atracción. En estos casos, especialmente en el primero de ellos, el CMA-ES se comporta mejor. Por supuesto, esto es una generalización. Es posible obtener más conclusiones si se realiza un análisis más profundo.

Comparando el rendimiento de los cuatro algoritmos en funciones de tipo *L-Separables* y unimodales, todos ellos resuelven todos los casos presentados en este procedimiento de caracterización. En este tipo de funciones, las direcciones en las que se realice la búsqueda no son relevantes, ya que a partir de cualquier punto del espacio de búsqueda es posible trazar un camino al óptimo ya que no existen óptimos locales. Si se analiza el CPEM en detalle, el algoritmo CMA-ES es el que mejor resultados en cuanto a velocidad de convergencia obtiene, seguido del DE y el RCGA. Finalmente, el MA es el algoritmo más lento en este tipo de funciones (un valor bajo de CPEM indica que el óptimo se ha alcanzado en un número menor de FEs, es decir, que el algoritmo es más rápido).

Lo resultados obtenidos por los cuatro algoritmos en las funciones *NL-Separable* y unimodales, muestran que el DE es el algoritmo que, en general, mejor se comporta, resolviendo todos los casos presentados. De hecho, es el único algoritmo que resuelve la función *Schwefel 2.21* con 30 y 50 dimensiones. Como se comentó en secciones anteriores, esta función tiene un camino al óptimo muy largo, lo cual provoca que el espacio de calidad sea muy complejo. En el caso particular de la función *Easom*, todos los algoritmos se comportan, más o menos, de manera satisfactoria, aunque el CMA-ES es el que obtiene los peores resultados. Esta es una función de dos dimensiones con un espacio de calidad particular, ya que presenta un área plana muy amplia (aproximadamente el 90% del espacio de calidad) con un óptimo local situado aproximadamente en el centro del mismo, así que, durante la búsqueda, cuando la población se sitúa sobre la superficie plana, el CMA-ES no tiene suficiente información para crear la matriz de covarianzas de manera adecuada.

Las últimas funciones unimodales consideradas en este conjunto de funciones prueba son *No-Separables*. Como muestran las tablas de resultados, el DE es la apuesta más segura en funciones de baja dimensionalidad (el CMA-ES falla en algunas ejecuciones de la función Kowaliks) debido a que es el más robusto cuando se enfrenta a espacios de calidad con características topológicas complejas como largos caminos hacia el óptimo, mientras que el CMA-ES es mejor con funciones de altas dimensiones. De hecho, uno de los puntos débiles del DE son las funciones no separables de alta dimensionalidad. Es interesante destacar que ninguno de los algoritmos resuelve todas las ejecuciones de la función *Perm*. Esta función, como ya fue comentado anteriormente, presenta el camino más largo hasta el óptimo (ver tabla 4.5).

Comparando el rendimiento de los tres algoritmos sobre funciones multimodales, y teniendo en cuenta las funciones separables (tanto las linealmente separables como las no linealmente separables), el DE es claramente el mejor algoritmo sin tener en cuenta el tamaño de los centros de atracción y la distancia entre ellos. El RCGA proporciona, en general, resultados satisfactorios, mientras que el CMA-ES y el MA fallan cuando la función es de alta dimensionalidad. En este sentido, la respuesta del CMA-ES ante este tipo de funciones es la más inestable y, por lo tanto, es el algoritmo menos recomendable. El MA mejora los resultados del CMA-ES cuando el tamaño del centro de atracción óptimo no es muy pequeño (veáse la función *Cosine*).

Finalmente, para las funciones multimodales y *No-Separables*, tanto el DE como el CMA-

ES pueden ser recomendados (el RCGA obtiene los peores resultados). El algoritmo MA se recomienda cuando se trata de funciones de baja dimensionalidad, aunque sus resultados en cuanto a rendimiento son peores que los obtenidos tanto por el DE como por el CMA-ES. Si se tienen en cuenta las funciones resueltas, el MA resuelve las mismas funciones que el DE. Los tres fallan en funciones *No-Separables* de alta modalidad como la función *Rosenbrock* o la función *Griewank*. Funciones como la *Rosenbrock* que presentan de manera conjunta alta dimensionalidad, no separabilidad y multimodalidad, son las funciones más complicadas para todos los algoritmos, aunque parece que la no separabilidad es un problema mayor para el DE que la multimodalidad para el CMA-ES, al menos en este caso. Además, por un lado, el DE también tiene problemas con funciones no separables de baja dimensionalidad cuando hay pocos centros de atracción (como en la función *Hartman 6*). Esta característica parece no afectar al CMA-ES, al menos si la distancia entre estos centros de atracción es corta. Por otro lado, el CMA-ES también tiene problemas con ciertas características topológicas de los espacios de calidad de funciones multimodales, incluso cuando la dimensionalidad no es alta, como largas distancias entre centros de atracción o cuando el centro de atracción óptimo no es el más amplio, como en la función *Shekel 5*.

En este capítulo se ha aplicado el procedimiento de caracterización presentado en este trabajo sobre tres algoritmos muy utilizados en el campo de la CE, tanto en problemas de optimización matemáticos como en problemas de aplicaciones del "mundo real", y sobre un cuarto algoritmo más reciente. Ha quedado demostrada la gran importancia que tiene realizar una caracterización formal como la propuesta, ya que se han extraído conclusiones sobre el rendimiento de los algoritmos en función de la separabilidad y modalidad del espacio de calidad.

Estas conclusiones han permitido conocer qué propiedades de los algoritmos son intrínsecas a sus estrategias de búsqueda y cuáles pueden ser modificadas parcialmente variando sus parámetros.

Capítulo 6

Conclusiones y principales aportaciones

El principal objetivo de esta tesis doctoral ha sido el desarrollo de un procedimiento de caracterización de algoritmos evolutivos. Este procedimiento permite la realización de un análisis formal de la respuesta de los algoritmos en función de las características de los espacios de calidad que se seleccionen como más representativas en el tipo de problemas a tratar.

Para lograr este objetivo global, en primer lugar, se ha realizado una revisión bibliográfica de los trabajos existentes en el campo de la Computación Evolutiva dedicados al planteamiento y comparación de algoritmos evolutivos, tal y como se muestra en la sección 3.2 del capítulo 3. Esta revisión fue dividida en dos sub-secciones con el fin de presentar la problemática desde dos puntos de vista diferentes: el punto de vista del usuario de AEs (sección 3.2.1) y el punto de vista del desarrollador o diseñador de AEs (sección 3.2.2). Las conclusiones obtenidas después de esta revisión sugieren que, a pesar del uso extendido de los AEs, los trabajos que tratan sobre la presentación y análisis de este tipo de técnicas de optimización carecen de una metodología formal y estándar. Este hecho dificulta el uso de los AEs por parte de usuarios de otros campos diferentes al de la CE. Por tanto, se detecta la necesidad de realizar esfuerzos en esta línea de estandarización que ayuden a los desarrolladores de algoritmos a presentar sus resultados de una forma objetiva que permita a usuarios de otros campos conocer a priori si un determinado AE es adecuado para sus problemas o no. En la sección 3.2.2, también se presentó el resultado de una profunda revisión de la bibliografía dedicada a solventar este aspecto de falta de formalización en la caracterización de los algoritmos. La conclusión obtenida ha sido que los autores han tratado tres grandes tipos de problemas, que parten de las siguientes evidencias:

- Los conjuntos de funciones de prueba utilizados para caracterizar el comportamiento de los AEs no son adecuados debido a que no se caracterizan de forma realmente útil. La clasificación utilizada sigue siendo binaria, indicando si una función tiene o no una determinada característica pero sin analizarla detalladamente. No existe un análisis más profundo de las funciones de prueba que se pueda utilizar a la hora de estudiar los AEs, de forma que las conclusiones obtenidas sean útiles desde un punto de vista práctico.

- No existe un consenso en cuánto a qué medidas de error o rendimiento utilizar para analizar el comportamiento de los algoritmos y como utilizar los datos obtenidos para obtener conclusiones que contengan información práctica.

- En cuanto al análisis y caracterización de los algoritmos, se concluye que, antes de analizar el comportamiento de un algoritmo, es necesario establecer los objetivos del

propio análisis para elegir de manera adecuada las medidas que se van a utilizar.

Por lo tanto, partiendo de esta división de tareas básicas, el procedimiento de caracterización que se ha planteado consta de tres etapas que afrontan cada una de ellas y propone soluciones particulares para el caso de los problemas de optimización con parámetros reales.

Estas tres etapas se desarrollaron en el capítulo 4. En una primera sección se abordó el problema de la correcta elección y caracterización de los conjuntos de funciones prueba. Para abordar dicho problema, en la sección 4.1 se ha descrito detalladamente el concepto de espacio de calidad, estableciendo con claridad la relación entre las funciones de prueba y los espacios de calidad que generan. De la revisión realizada en la sección 3.2 se concluyó que la caracterización que se realiza actualmente de las funciones objetivo es insuficiente y no proporciona información realmente útil sobre las mismas. Con el fin de mejorar esta caracterización, se realizó un análisis de las propiedades principales de estos espacios para concluir que las más significativas son, en una primera aproximación, la modalidad y la separabilidad. Además, se ha realizado una breve revisión de trabajos anteriores donde los autores tratan de estimar la dificultad de un espacio de calidad teniendo en cuenta estas dos propiedades. A la vista de los trabajos encontrados y analizados se llegó a la conclusión de la necesidad del desarrollo de nuevos algoritmos de estimación de dichas características, debido a que las medidas que se han utilizado hasta este momento se centraron en el análisis de la dificultad para un algoritmo concreto, generalmente el AG clásico, por lo que no pueden generalizarse a otros paradigmas de computación evolutiva. En la sección 4.1.3 se desarrollaron estos algoritmos.

El primero de los algoritmos desarrollados sirve para analizar los espacios de calidad en términos de separabilidad. Como hemos dicho, del análisis del concepto de espacio de calidad se concluyó que la separabilidad es una de las características que más afectan a los AEs. El término separabilidad está relacionado con el concepto biológico de epístasis, que mide las relaciones que existen entre los genes de los cromosomas. Si el nivel de epístasis es alto, los genes de un cromosoma están muy relacionados y cambios en uno de ellos afectan a otros genes del cromosoma. Estimar el nivel de separabilidad o epístasis de un espacio de calidad es útil a la hora de decidir qué estrategia de búsqueda es más adecuada. Este algoritmo se ha validado sobre el conjunto de funciones de prueba seleccionado para esta tesis obteniendo los resultados que se muestran en la sección 4.1.3. Tras la aplicación de este algoritmo, las funciones consideradas han sido clasificadas en tres subconjuntos dependiendo de su separabilidad, así se obtienen:

- **Funciones L-Separales:** son funciones entre cuyas variables no existe ningún tipo de dependencia. Debido a esto, dichas funciones pueden separarse en n funciones de una variable independientes y optimizar cada una de estas funciones por separado.

- **Funciones NL-Separables:** son funciones entre cuyas variables existen dependencias de tipo no lineal. Estas dependencias provocan que varíe la "forma" del espacio de calidad dependiendo del valor que tomen las variables. A pesar de que en este tipo de funciones el óptimo siempre se sitúa en la misma posición sin importar el valor que tomen las variables, cuando se abordan con AEs que paralelizan el proceso de búsqueda modificando varias variables en cada generación, al presentar el espacio de calidad una topología variable, estos presentan problemas en el proceso de búsqueda.

- **Funciones No-Separables:** en este tipo de funciones existen dependencias entre variables que provocan tanto modificaciones en cuanto a la topología del espacio como a la posición en la que se sitúa el óptimo de la función.

Como se demuestra en dicha sección 4.1.3, el algoritmo ha resultado útil para analizar espacios de calidad en términos de separabilidad y clasificar las funciones utilizando más información de la que se disponía inicialmente y a la que actualmente tienen acceso los usuarios.

En segundo lugar, se desarrolló el algoritmo de análisis de modalidad. A partir de la revisión realizada de trabajos que estudian la modalidad de los espacios de calidad, se concluyó que la información realmente relevante no es tanto el número de óptimos locales o globales que estos tienen como los centros de atracción de dichos óptimos locales, su tamaño y su distribución sobre el espacio de calidad. El algoritmo desarrollado en la sección 4.1.3 analiza los espacios de calidad en función de estos centros de atracción. Tras la validación del algoritmo sobre el conjunto de funciones prueba, surgió la necesidad del desarrollo de un tercer algoritmo que permitiese estimar la dificultad de las funciones unimodales en función de la longitud del camino hasta el óptimo. El desarrollo y validación de este algoritmo se presenta también en esta sección. Los resultados aportados por este algoritmo permiten estimar la topología de un espacio de calidad n-dimensional desconocido. De las posibles propiedades que se pueden estimar, como una primera aproximación se han propuesto las siguientes:

- Si la función es multimodal, para estimar la topología del espacio de calidad se consideran:
  - El número de centros de atracción que encuentra el algoritmo de caracterización y que representa una estimación del número de óptimos locales.
  - La distribución de los tamaños de los centros de atracción. La tendencia que siguen los tamaños de los centros de atracción dependiendo de la distancia de dichos centros al óptimo es una medida que ha permitido estudiar qué tipo de estrategia de búsqueda es más adecuada para cada distribución.
  - La distancia entre centros de atracción. En esta tesis se han considerado dos distancias:
    * La distancia máxima entre centros de atracción, como estimación de la concentración de los centros de atracción sobre el espacio de calidad.
    * La distancia entre el centro de atracción óptimo y el centro de atracción de mayor tamaño. Si estos centros de atracción no son el mismo, es decir, cuando esta distancia es mayor que cero, los espacios de calidad son más difíciles de resolver que en el caso contrario, ya que los AEs suele tender hacia los centros de atracción de mayor tamaño.
- Si la función es unimodal, la mayor o menor dificultad de una función de este tipo depende de la longitud máxima del camino hasta el óptimo desde los extremos del espacio de calidad.

La información acerca de los espacios de calidad que se obtiene después de la aplicación de estos tres algoritmos en términos de separabilidad y modalidad será básica a la hora de caracterizar formalmente los AEs, ya que el análisis de su comportamiento se hará en base a su respuesta en dichos espacios de calidad. Como quedó demostrado en el capítulo de aplicación, la información detallada que proporcionan es realmente útil para conocer el ámbito de aplicación de un cierto algoritmo en un problema. Es cierto que el coste computacional de estos algoritmos, sobre todo el de análisis de modalidad, es elevado; pero dicho análisis solamente se realiza una vez sobre el conjunto de funciones de prueba y las conclusiones que permite obtener acerca de los algoritmos estudiados lo justifican.

La segunda etapa del procedimiento de caracterización está relacionada con las medidas de error y rendimiento consideradas para el análisis de AEs. En la sección 4.2 se ha realizado una revisión de las más utilizadas en los trabajos del campo concluyendo que no existe consenso entre los autores y, al existir varias, su uso complica la realización de un análisis adecuado. En esta sección se propone el uso de una nueva medida, *CPEM*, que combina información sobre error y rendimiento en un solo dato, favoreciendo así el análisis de la gran cantidad de datos generados. Esta medida es complementada en la sección de aplicación con el SR que da información cuantitativa sobre el número de ejecuciones falladas o acertadas. Dado que en este trabajo se propone el promedio de la ejecución de 25 pruebas como resultado base, esta medida resulta útil a la hora de realizar un análisis inicial de las funciones que presentan problemas a los AEs. Además, se propone también el uso del G_{NE}, una medida exclusiva de este tipo de algoritmos y que proporciona información muy relevante desde el punto de vista de la precisión del resultado obtenido. Como conclusión a esta etapa se puede decir que el procedimiento de caracterización propone el uso de tres medidas de error para analizar los datos, una básica, el *CPEM*, y otras dos para obtener información adicional, el SR y el G_{NE}.

Por último, en la revisión realizada al inicio de esta tesis se observó que no existe un consenso generalizado ni en la forma en la que se realizan las pruebas para presentar algoritmos ni en la forma en la que se analizan los resultados obtenidos. Es decir, cada autor caracteriza su algoritmo de manera diferente a otros haciendo dificil la comparación de resultados con otros algoritmos diferentes. Los intentos más serios en este sentido se basan en el diseño de experimentos, pero estos trabajos se centran en la problemática de estudiar un algoritmo en la fase de diseño, realizando un barrido paramétrico estipulado y estudiando la respuesta. Este procedimiento está fuera del alcance de esta tesis, que parte del hecho de que el algoritmo a caracterizar ya ha sido desarrollado y se conoce su funcionamiento básico.

La tercera etapa del procedimento de caracterización afronta este problema estableciendo una serie de condiciones de ejecución objetivas de forma que los resultados de los algoritmos analizados sean comparables y planteando una metodología de análisis de los resultados. En cuanto a las condiciones de ejecución, éstas se presentan en la sección 4.3.1 y se basan en las utilizadas en la mayor parte de las competiciones sobre AEs que se celebran en la actualidad en los congresos del área:

- El desarrollador deberá indicar todos los parámetros de configuración excepto el tamaño de población. Éste se obtendrá tras un análisis exhaustivo en cada una de las funciones analizadas.
- Para cada algoritmo, función y dimensión se ejecutarán 25 pruebas independientes.
- El criterio de parada utilizado se basa en el número de llamadas a la función de calidad y depende de la dimensión del problema. Concretamente, el número máximo de llamadas permitidas es $10000 \cdot n$, siendo n la dimensión del problema.
- Se utilizarán como medidas de error y rendimiento el CPEM, el SR y el G_{NE}.
- Se considera que una prueba está resuelta cuando su valor de error absoluto es menor que 10^{-6}.

A la hora de analizar los resultados obtenidos en las pruebas, se propone una metodología basada en tres pasos consecutivos: análisis de resultados con la configuración básica, variando los parámetros iniciales y comparativamente. En todos los casos, el análisis se debe

llevar a cabo en función de las características de los espacios de calidad seleccionadas, en este caso, separabilidad y modalidad. Las conclusiones deben ser extraídas en función de las propiedades básicas de los AEs, que como se muestra en la sección 4.3.2 son: balance exploración / explotación, presión selectiva, direcciones de búsqueda y paso de mutación. Cada una de ellas ha sido descrita en detalle y se ha explicado como influyen en las estrategias de búsqueda de los AEs. La configuración de los operadores de los AEs es la que determina el peso de cada una de estas propiedades.

Resumiendo, en el desarrollo del capítulo 4 de esta tesis se ha propuesto un procedimiento de caracterización formal de AEs que debe afrontar tres grandes problemas, para cada uno de los cuales se proponen posibles soluciones prácticas que han sido validadas en el capítulo 5.

Así, con el objetivo de demostrar que el procedimiento de caracterización desarrollado utilizando las soluciones propuestas permitirá analizar AEs de forma que se aporten conclusiones formales relevantes que contengan información útil para los usuarios, en el capítulo 5 se han analizado cuatro AEs que utilizan estrategias de búsqueda muy diferentes. Estos cuatro algoritmos son: un algoritmos genéticos de codificación real (RCGA), el algoritmo Covariance Matrix Adaptation (CMA-ES), el algoritmo Differential Evolution (DE) y los Algoritmos Macroevolutivos (MA). Los tres primeros algoritmos han sido seleccionados por ser altamente representativos en el campo actual de la CE. El último es un algoritmo menos conocido y, por lo tanto, menos utilizado. Sin embargo, su caracterización es de gran interés debido a lo diferente de su filosofía con los tres algoritmos restantes. Para llevar a cabo la caracterización se ha utilizado el conjunto de funciones de prueba que fue estudiado en la sección 4.1. Con esta información, y con los resultados obtenidos tras la ejecución de los algoritmos en las condiciones establecidas en la sección 4.3.1, se ha analizado en detalle el comportamiento de cada uno de ellos, relacionando dicho comportamiento con las características observadas en las funciones utilizadas en el conjunto de funciones de prueba, en este caso, separabilidad y modalidad. Como resultado de este análisis se han obtenido conclusiones con información acerca de cuales son los espacios de calidad más idóneos para cada algoritmo. Además, a partir de las conclusiones obtenidas tras un primer análisis, ha sido posible realizar un ajuste de los algoritmos de cara a comprobar si se mejoran sus resultados o si el fallo es intrínseco a su diseño. Este ajuste no se ha basado en un procedimiento de prueba y error, sino que, utilizando la información obtenida mediante el procedimiento de caracterización, ha sido posible guiarlo en la dirección adecuada. Si bien es cierto que, como postula el teorema *No-Free Lunch*, no existe ningún algoritmo que sea el mejor en todo tipo de funciones, sí que es posible mejorar los resultados de un cierto AE si se conoce el porqué de su comportamiento y cómo afectan sus parámetros a su modo de búsqueda.

Por tanto, la conclusión final al trabajo desarrollado en esta tesis doctoral es que aporta un primer intento en la caracterización de algoritmos evolutivos que proporcione un mayor grado de formalización al campo, lo que repercutirá en una mayor aplicabilidad en otros ámbitos. Los usuarios de algoritmos evolutivos que no pertenecen directamente al campo de la computación evolutiva dispondrán de algoritmos caracterizados en términos de para qué problemas son más adecuados. De esta forma, si un usuario conoce o puede estimar las características topológicas de su problema concreto y dispone de varios algoritmos evolutivos caracterizados siguiendo este procedimiento, dispondrá de información suficiente para descartar entre uno u otro algoritmo.

Capítulo 7

Trabajo futuro

El trabajo realizado en esta tesis ha permitido abrir una línea de investigación que continuará su desarrollo en los próximos años con diversas sub-líneas que parten de la principal.

A partir del trabajo desarrollado en cuanto a análisis y caracterización de espacios de calidad se tratará, en primer lugar, de añadir más funciones al conjunto de funciones de prueba. Si bien es cierto que el conjunto utilizado incluye un amplio rango de funciones con diferentes topologías, resultado de aglutinar las utilizadas en los trabajos presentes en la bibliografía del campo, no estaría de más incorporar funciones adicionales al conjunto de prueba, aumentando el número de funciones con determinadas características y, por tanto, aumentando la representatividad de dicho conjunto de prueba. Esto parece particularmente necesario para las funciones multimodales, no separables y de alta dimensionalidad. En segundo lugar, se pretende extender el dominio de aplicación de este trabajo a problemas con restricciones o problemas multiobjetivo, por lo que se deberán analizar de nuevo los espacios de calidad relacionados y desarrollar nuevos algoritmos para conocer de forma aproximada sus características. Otro ámbito de aplicación que se desea abordar, dada su relevancia dentro de la investigación que se lleva a cabo en el Grupo Integrado de Ingeniería, es el de la evolución de redes de neuronas artificiales. Esto requerirá también la creación de otro conjunto de funciones de prueba que incluya problemas dinámicos.

En esta tesis las propiedades de separabilidad y modalidad han sido seleccionadas como las principales para el desarrollo del método de caracterización de espacios de calidad. En este sentido, se profundizará en el estudio de las propiedades de estos espacios relevantes para el análisis del comportamiento de los algoritmos evolutivos, ya sea incorporando nuevas propiedades al análisis o profundizando en las dos ya utilizadas, especialmente en la modalidad, explorando conceptos tales como densidad de centros de atracción, poder de atracción, etc.

Relacionado con los dos puntos anteriores (necesidad de aumentar el número de funciones de prueba y de profundizar en el análisis de la modalidad), se desarrollará un generador de funciones de prueba configurable que permita modificar la topología de una función en términos de la distribución de sus óptimos (locales y globales). El alcance de esta tesis ha incluido una primera versión de su desarrollo, pero no se ha terminado completamente, por lo que queda propuesto como trabajo futuro. Este generador de funciones de prueba permitiría analizar el impacto de la topología de los espacios de calidad sobre los algoritmos de una forma más exhaustiva al poder generar cualquier combinación topológica y no depender de las funciones del conjunto de prueba. Además de estudiar el efecto del tamaño y la distribución de los centros de atracción, este generador debería permitir el análisis en términos de poder de atracción de una cuenca, entendiendo el poder de atracción como el diámetro y la

altura de cada centro de atracción.

Adicionalmente, el algoritmo de análisis de separabilidad deberá ser mejorado para proporcionar algún tipo de medida cuantitativa, y no sólo cualitativa, en cuanto a la presencia de separabilidad en una función dada.

Se tratará también de crear versiones específicas de los algoritmos de análisis de modalidad y separabilidad para ser usados por usuarios de algoritmos evolutivos, de forma que su coste computacional sea menor y puedan ser aplicados no ya para determinar las características topológicas de las funciones de prueba, si no para estimar dichas características en problemas reales.

En cuanto al análisis del comportamiento de los algoritmos, se pretende ampliar el número de ellos analizados generando, de esta forma, una fuente de conocimiento acerca de los mismos que sea accesible y útil para todos los usuarios. Como parte de la línea de investigación abierta, y a partir de los análisis realizados a los algoritmos considerados en esta tesis, se pretenden realizar modificaciones a dichos algoritmos con el objetivo de mejorar sus puntos débiles guiándonos por las conclusiones obtenidas en cuanto a sus estrategias de búsqueda, especialmente en el caso del MA, algoritmo original en cuanto a concepción y todavía poco explotado.

Además de todos estos puntos extraídos del desarrollo propio de la tesis, continuando con el desarrollo de la librería JEAF, que surgió como parte de esta tesis, se desarrollarán herramientas de análisis de espacios de calidad que sean "amigables" para los usuarios tanto en su aplicación como en la forma de presentar los resultados obtenidos.

Apéndice A

JEAF : A Java Evolutionary Algorithm Framework

En el campo de la computación evolutiva no existe muchas herramientas software que permitan a los investigadores implementar, modificar o comparar diferentes algoritmos de forma sencilla. Las pocas que existen normalmente carecen de flexibilidad, de mantenimiento u otras características básicas, así que los investigadores acaban programando sus propias soluciones, muchas veces reimplementando algoritmos que han sido implementados cientos de veces.

En este apéndice se presenta una nueva librería para computación evolutiva llamada Java Evolutionary Algorithm Framework (JEAF) que trata de ofrecer una plataforma que facilite las tareas de comparar, analizar, modificar e implementar algoritmos evolutivos reutilizando componentes y realizando el mínimo esfuerzo en cuanto a codificación. Además JEAF trata de ser una herramienta para los usuarios de algoritmos evolutivos en áreas diferentes a las ciencias de la computación, tales como Ingeniería, Biología, Física aplicada, etc. En este sentido, JEAF proporciona métodos para la distribución del proceso evolutivo, reduciendo así su coste computacional, y para el uso de herramientas externas en el proceso de evaluación, lo cual permite aprovechar las ventajas de otras aplicaciones del campo de aplicación del problema concreto a resolver tales como simuladores hidrodinámicos u otras herramientas de cálculo utilizadas en campos como la Ingeniería. Esta librería ha sido desarrollada durante la realización de esta tesis doctoral y ha dado soporte al procedimiento de caracterización propuesto.

A.1 Introducción

Cuando un investigador quiere implementar un nuevo AE o trata de resolver un problema particular con diferentes AEs, generalmente se encuentra con el mismo problema: la falta de una herramienta software actualizada, flexible y completa que permita realizar dicha tarea con un mínimo esfuerzo. Existen herramientas que implementan AEs, pero suelen presentar problemas como, por ejemplo:

- Pocos algoritmos implementados.
- Herramientas desactualizadas o poco mantenidas que, por lo tanto, no implementan algoritmos recientes o tienen problemas con compiladores actuales.

- Las implementaciones de AEs que ofrecen generalmente no están validadas frente a los conjuntos de funciones de prueba de referencia.
- Poca flexibilidad, requiriendo un gran conocimiento de la herramienta para modificar las implementaciones, llegando incluso a requerir el mismo esfuerzo que si la implementación se hiciese desde cero.
- Falta de documentación.

Por lo tanto, la situación más común es que los investigadores terminen implementando los algoritmos desde cero, contribuyendo a que este problema de falta de estandarización sea mayor. Es decir, haciendo que esta nueva implementación sea difícil de comparar, mejorar, adaptar y utilizar por otras personas. El objetivo de JEAF es el de proporcionar una herramienta con una amplia y probada colección de AEs, un conjunto de funciones de prueba típicos y un diseño que facilite la implementación de nuevos algoritmos, haciendo que el tiempo necesario para implementarlos sea lo menor posible.

Se ha escogido JAVA como lenguaje de programación debido a que uno de los requisitos principales era el de ofrecer la posibilidad de ejecutar la herramienta en cualquier plataforma sin necesidad de recompilado (sobre todo teniendo en cuenta su ejecución en un cluster). Sin embargo, también ha sido desarrollada con el objetivo de hacerla lo más rápida posible desde el punto de vista computacional. Los AEs son utilizados normalmente para resolver problemas complejos, así que esta herramienta proporciona posibilidades de distribución para acelerar las evoluciones y ha sido desarrollada para que el uso de herramientas externas en la fase de evaluación sea sencillo.

En la bibliografía existen algunas herramientas representativas en el área de los AEs, algunas de ellas, aunque son conocidas carecen de algunos elementos esenciales o actualmente no continúan su desarrollo. Estas herramientas son:

- PGAPack [PGA, 2009] es probablemente la primera herramienta pública para computación evolutiva. Se encuentra accesible desde Febrero de 1996, pero no se actualiza desde Junio del 2009 y actualmente su página web no se encuentra accesible. Fue escrita en C y contiene solamente algoritmos genéticos.
- GAUL [GAU, 2009] está escrita en C y centrada en algoritmos genéticos, aunque incluye una implementación del Differential Evolution y modelos distribuidos / paralelos. La última versión es de Octubre del 2006, aunque existen aportaciones en su versión CVS hasta Marzo del 2009.
- TEA [TEA, 2003] implementa algoritmos genéticos, estrategias evolutivas y algunos algoritmos multiobjetivo, está programada en C++ y soporta algunos modelos de distribución para evaluación. La última versión es de Mayo del 2003.
- Open BEAGLE [BEA, 2009] es otra buena herramienta de computación evolutiva desarrollada en C++, comparte algunos de los objetivos de este trabajo, sin embargo está implementada en un lenguaje de programación diferente con sus ventajas e inconvenientes. Su desarrollo es muy intermitente y la última versión estable es de Diciembre del 2007, aunque la ultima versión desarrollada (alpha, CVS) es de Febrero del 2009.
- JDEAL [JDE, 1999] está escrita en JAVA, pero su desarrollo terminó en Diciembre de 1999. Implementa algoritmos genéticos y estrategias evolutivas y soporta el modelo de distribución maestro - esclavo.

- OpenOpal [Ope, 2010] es otra herramienta JAVA desarrollada para aplicaciones de optimización automática y aprendizaje máquina. Esta herramienta soporta problemas mono-objetivo y multi-objetivo, pero solamente implementa una estrategia evolutiva y el algoritmo CMA-ES.

- JMetal [jMe, 2006, Durillo et al., 2006] es una herramienta muy completa centrada en metaheurísticas multiobjetivo. Permite el desarrollo de nuevas metaheurísticas utilizando clases que funcionan como bloques constructivos. Proporciona una serie de implementaciones de algoritmos multiobjetivo muy conocidos.

- La herramienta más parecida a JEAF es ECJ [ECJ, 2010], un entorno excelente para computación evolutiva también programada en JAVA. Sin embargo ECJ es menos flexible que JEAF y además es más lenta cuando se trabaja con cromosomas largos en cuanto a número de genes. Hay que hacer énfasis en que JEAF, además de tener implementados un gran número de algoritmos, proporciona también un conjunto amplio de funciones de prueba y herramientas de análisis con el objetivo de proporcionar una solución completa que minimice los esfuerzos por parte de los investigadores de computación evolutiva y los investigadores de otros campos.

Para el desarrollo de JEAF se han tenido en cuenta las ventajas y desventajas de cada unas de las librerías que han sido comentadas. El principal objetivo de JEAF es el desarrollo de una herramienta software que facilite la estandarización tanto en el desarrollo de nuevos AE como en la comparación de estos. JEAF se ha desarrollado teniendo en cuenta dos puntos de vista diferentes: el punto de vista del investigador en CE o desarrollador de algoritmos y el punto de vista del investigador de otras áreas o usuario de AE. Desde el punto de vista del desarrollador, el objetivo de JEAF ha sido implementar una herramienta que permita de manera sencilla desarrollar nuevos algoritmos o versiones de algoritmos existentes. Desde el punto de vista del usuario, teniendo en cuenta que en la mayoría de las ocasiones su conocimiento acerca de los AE es reducido, JEAF le permitirá trabajar con AEs sin conocer exactamente el funcionamiento de los mismos.

A.2 Principales características

Esta sección contiene un resumen de las principales características de JEAF, las cuales están directamente relacionadas con los objetivos de diseño establecidos para su desarrollo:

- **Multiplataforma:** JEAF está implementada en JAVA, así que puede utilizarse en cualquier plataforma hardware donde esté instalada una máquina virtual JAVA. La versión requerida es la 1.5 o superior ya que se utilizan algunas características recientes como los tipos genéricos.

- **Modular:** el diseño se ha realizado profundamente orientado a objetos, tratando de mantener la mayor independencia posible entre componentes.

- **Flexible:** esta característica está relacionada con la anterior y está presente en cada pieza de la librería. JEAF fue diseñada no sólo para dar soporte a los AEs actuales, sino también para permitir el uso de cualquier combinación de operadores que el usuario necesite para su problema. De esta forma, se pueden implementar futuros AEs sin

necesidad de que estos deban encajar en la estructura clásica de un AE y sin reimplementar componentes que ya existan. De hecho, la librería facilita la reutilización de componentes, dividiendo el comportamiento general de un AE en cuatro etapas (selección, reproducción, evaluación y reemplazo) e implementando cada etapa con una cadena de operadores de longitud arbitraria. Lo más común son longitudes de uno o dos operadores, pero cualquier valor entero positivo e incluso cero está permitido, siempre y cuando se respeten las interfaces. Además, muchas de las características comunes de los operadores están implementadas en torno a *plug-ins* y son fácilmente intercambiables. Esto permite la implementación de un mecanismo más potente que una lista prefijada de valores.

- **Eficiencia:** desde el comienzo del desarrollo de JEAF, la eficiencia ha sido uno de los mayores retos. Los AEs consumen mucho tiempo de cómputo, especialmente cuando son aplicados a problemas reales, y la elección del lenguaje JAVA tiene un impacto negativo si lo comparamos con lenguajes como C o C++. Así que algunas decisiones han sido tomadas con mucho cuidado para maximizar la eficiencia de la implementación.

- **Fácil de usar:** teniendo en mente que JEAF va a ser utilizada por investigadores que no pertenecen al área de la computación evolutiva, fue diseñada para que los usuarios no necesitasen conocer el comportamiento exacto de un AE, simplemente conociendo el comportamiento global y los elementos de la librería que tienen que personalizar para resolver un problema particular.

- **Herramientas de análisis:** además de diferentes AEs y funciones de prueba, esta librería proporciona herramientas de análisis, además de permitir la implementación de otras nuevas. Estas herramientas han sido desarrolladas con el objetivo de facilitar la comparación de resultados obtenidos aplicando distintos AEs o diferentes versiones de un mismo algoritmo. Se incluyen herramientas de análisis de AEs y herramientas de análisis de funciones.

- **Soporte para procesado distribuido:** JEAF soporta un modelo muy flexible de distribución para el proceso evolutivo. Utiliza MPI como protocolo de comunicación (la decisión fue tomada, de nuevo, por temas de eficiencia). Se soportan varios modelos (maestro-esclavo, islas, etc.) con distintas topologías (rejilla, anillo, etc.) y formas de intercambiar el material genético. Los modelos soportados permiten también la combinación de los mismos para crear esquemas híbridos que se adapten a un problema particular o a la disponibilidad de hardware.

A.3 Diseño e implementación

Esta librería se ha dividido en tres módulos: el primero de ellos es el módulo de Algoritmos Evolutivos, implementa los elementos necesarios para crear y utilizar un AE. El segundo módulo es el de funciones de prueba, donde el usuario puede encontrar un conjunto amplio de las funciones más utilizadas. Por último, existe un módulo de análisis que implementa dos tipos de herramientas: para el análisis de algoritmos y para el análisis de funciones objetivo. Todos estos módulos se describen con detalle en las siguientes secciones.

A.3.1 Módulo de Algoritmos Evolutivos

Este es el componente principal de la librería. Es el encargado de implementar la estructura que da soporte al desarrollo de los algoritmos evolutivos, además de incluir algunas implementaciones de AEs concretos. El núcleo del módulo principal de esta librería se puede ver en la figura A.1. El elemento principal es la clase *EvolutionaryAlgorithm*, la cual constituye el centro de todas las operaciones y es la encargada de dirigir el comportamiento del algoritmo.

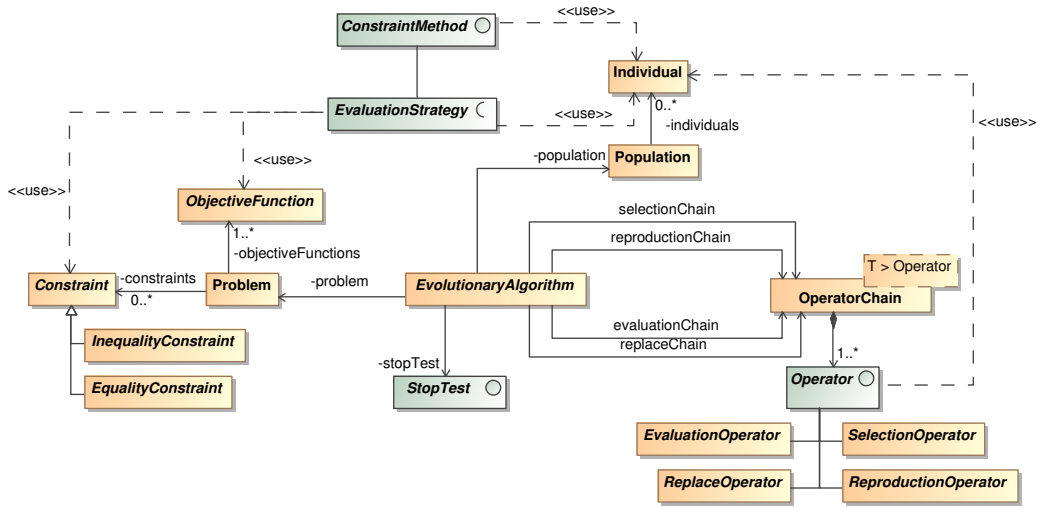


Figura A.1: Diagrama de clases del núcleo del módulo de algoritmos evolutivos.

Los AEs implementados con esta librería necesitan los siguientes elementos:

- Una población, formada por una lista de individuos.
- Cuatro cadenas de operadores para las fases de selección, reproducción, evaluación y reemplazo. Cada una de estas cadenas está formada por una lista de operadores que se ejecutan de forma sucesiva.
- Un problema para resolver que está definido por, al menos, un objetivo y una lista de funciones de restricción si fuera necesario.
- Una estrategia de evaluación encargada de evaluar a cada individuo con las funciones objetivo y las funciones de restricción, si existen. También es la encargada de dirigir el comportamiento del método de manejo de restricciones si es necesario.
- Un criterio de parada que determina cuándo finaliza el proceso de evolución de un algoritmo.

Todos estos componentes serán explicados en detalle en los siguientes apartados.

Población

En JEAF, una población se representa mediante una lista de individuos. El paquete *population* es el encargado de implementar la estructura de las poblaciones de los AEs en esta librería. La estructura de clases de este paquete se puede ver en la figura A.2.

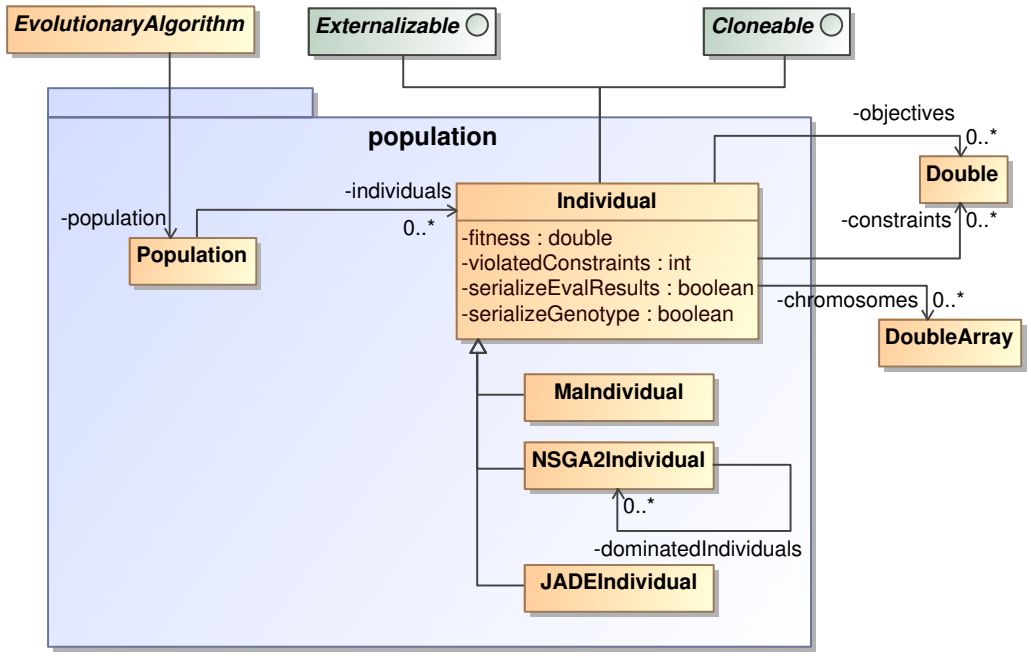


Figura A.2: Diagrama de clases del paquete *population* de la librería JEAF.

Además de la clase *Population*, el elemento clave de este paquete es la clase *Individual*, encargada de implementar a un individuo de la población. En las implementaciones básicas, un individuo representa una solución al problema y tiene los siguientes atributos:

- Una lista de cromosomas, *chromosomes*, uno o varios, que codifican la solución que representa el individuo. Los cromosomas se implementan como *arrays* de números de tipo *double* que pertenecen al rango [-1:1]. Las funciones objetivo son las encargadas de normalizar estos valores al rango correspondiente según el problema a resolver.
- Una lista de valores objetivos, *objective*, que representan los valores obtenidos al evaluar al individuo con cada una de las funciones objetivo del problema.
- Una lista de valores de restricciones, *constraints*, que representan los valores obtenidos al evaluar al individuo con cada una de las funciones de restricción del problema, si estas existen.
- El valor de calidad del individuo, representado por la variable *fitness*.
- El número de restricciones violadas, representado por la variable entera *violatedConstraints*.
- Un atributo booleano *serializeEvalResults* que toma el valor verdadero o falso según se serialicen los objetos *objectives* y *constraints* o no.
- Un atributo booleano *serializeGenotype* que, al igual que el atributo anterior, toma el valor verdadero o falso según se serialice o no, en este caso, la lista de cromosomas.

El individuo básico puede ser extendido con el fin de añadir más funcionalidades además de las que tiene implementadas por defecto. Actualmente, JEAF tiene implementados tres individuos para tres algoritmos concretos: el individuo *MAIndividual* para el algoritmo

macroevolutivo, el individuo *NSGA2Individual* para el algoritmo multiobjetivo NSGA2 y el individuo *JADEIndividual* para el algoritmo JADE. El diagrama de clases de la figura A.3 representa estas subclases de individuos específicos y en ella aparecen los atributos específicos de cada una de esas subclases.

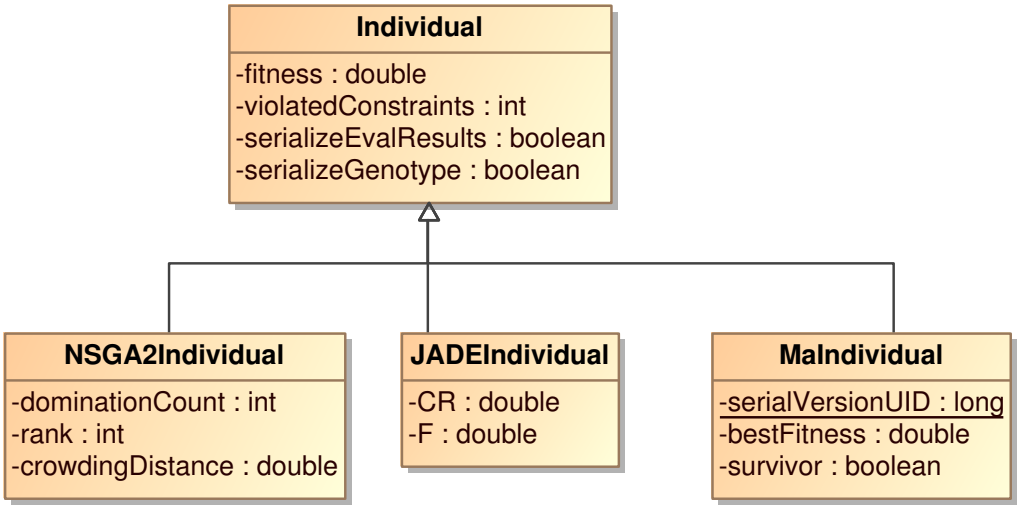


Figura A.3: Diagrama de clases de las subclases de la clase *Individual* implementadas actualmente en la librería JEAF.

La representación de los individuos como listas de cromosomas de valores *double* tiene dos ventajas principales:

1. Esta representación permite que los individuos sean independientes del problema a resolver y que, en principio, solamente las funciones objetivo sean dependientes del problema, ya que estas, como veremos en la sección A.3.1, son las encargadas de normalizar los valores de los cromosomas y realizar la transformación genotipo-fenotipo si fuese necesario.

2. La implementación del AE, y más concretamente de los operadores utilizados, es también independiente del problema a resolver. De esta forma podemos optimizar la implementación de los algoritmos sin preocuparnos de la representación de los individuos ya que siempre será la misma, aprovechando así implementaciones de operadores ya probadas.

Cadenas de operadores

La ejecución de un AE en JEAF está dividida en cuatro fases y en cada una de ellas se ejecutan diferentes operadores. Estas fases son: selección, reproducción, evaluación y reemplazo. Las cadenas de operadores que representan cada una de estas fases pueden contener uno o más operadores. Lo general son cadenas de uno o dos operadores, pero no se limita el número de estos. Las características de cada operador dependen de la fase en la que se está ejecutando:

- En la fase de selección se ejecutan operadores de tipo *SelectionOperator*. El comportamiento básico de estos operadores es el de seleccionar individuos que participarán en

la siguiente fase. Los operadores de esta cadena reciben una población de individuos y tras su ejecución devuelve una lista de copias de individuos seleccionados según el criterio implementado en el operador concreto.

- En la fase de reproducción, se generan nuevos individuos a partir de individuos de la población de padres. Los operadores que se ejecutan son del tipo *ReproductionOperator*. Actualmente existen dos subclases principales: los operadores de cruce, *CrossoverOperator*, y los operadores de mutación, *MutationOperator*. Los operadores de cruce implementan la reproducción sexual, es decir, partiendo de dos o más padres se genera uno o más hijos. La reproducción asexual se implementa en los operadores de mutación, en los cuales a partir de un padre se genera un hijo que es una copia del padre con algunas modificaciones.

- La siguiente fase es la de evaluación cuya cadena de operadores contiene operadores de tipo *EvaluationOperator*, esta cadena no es muy utilizada. En la fase de evaluación existen dos alternativas: utilizar una cadena de operadores o utilizar una estrategia de evaluación, las estrategias de evaluación se explicarán en detalle en la sección A.3.1.

- La última fase de un AE es la fase de reemplazo. En esta fase se ejecuta una cadena de operadores de tipo *ReplaceOperator*. El comportamiento típico de los operadores de este tipo es el de generar una población de tamaño N a partir de una población de padres (que proviene de la generación anterior) y la población de descendientes generada en la fase de reproducción.

En la figura A.4 se muestra el diagrama de clases del paquete *operator*. En este diagrama aparecen las cuatro clases principales de operadores: *SelectionOperator*, *ReproductionOperator* con sus subclases *CrossoverOperator* y *MutationOperator*, *EvaluateOperator* y *ReplaceOperator*. Estas clases implementan el patrón de diseño *plantilla*, en el cual se define una estructura de herencia donde la superclase sirve de plantilla de los métodos de la subclases. Las superclases implementan el método *operate* heredado de la interfaz *Operator*. Este es el método plantilla común para todas las subclases que hereden de cada superclase particular. Cada implementación concreta de la interfaz *Operator* implementa en el método *operate* el esqueleto general de comportamiento. Uno de los pasos de dicho esqueleto será implementado en los métodos *select*, *crossover*, *mutation*, *evaluate* y *replace*, según corresponda.

En el diagrama de clases de la figura A.5 se muestran los operadores que se encuentran implementados en la versión actual de la librería JEAF.

Todas las cadenas de operadores, sin importar el tipo concreto, poseen la misma estructura. Para todos los operadores de la cadena, la entrada al método principal *execute* es una lista de individuos que proviene o bien del algoritmo, cuando es el primer operador de la cadena, o bien es el resultado de la ejecución del operador anterior. Una vez ejecutados todos los operadores de la cadena, el algoritmo recibe una lista de individuos del último operador de la cadena. Este comportamiento se describe como diagrama de secuencia en la figura A.6.

Debido a la flexibilidad de este mecanismo, para la implementación de un algoritmo en JEAF no es necesario implementar una clase específica. La implementación de un nuevo algoritmo puede hacerse definiendo simplemente las cadenas de operadores precisas. De acuerdo con los AEs más típicos, JEAF incluye operadores y configuraciones para algoritmos genéticos, estrategias evolutivas, Differential Evolution, CMA-ES, algoritmos microgenéticos, algoritmos macroevoluctivos y el algoritmo NSGA2.

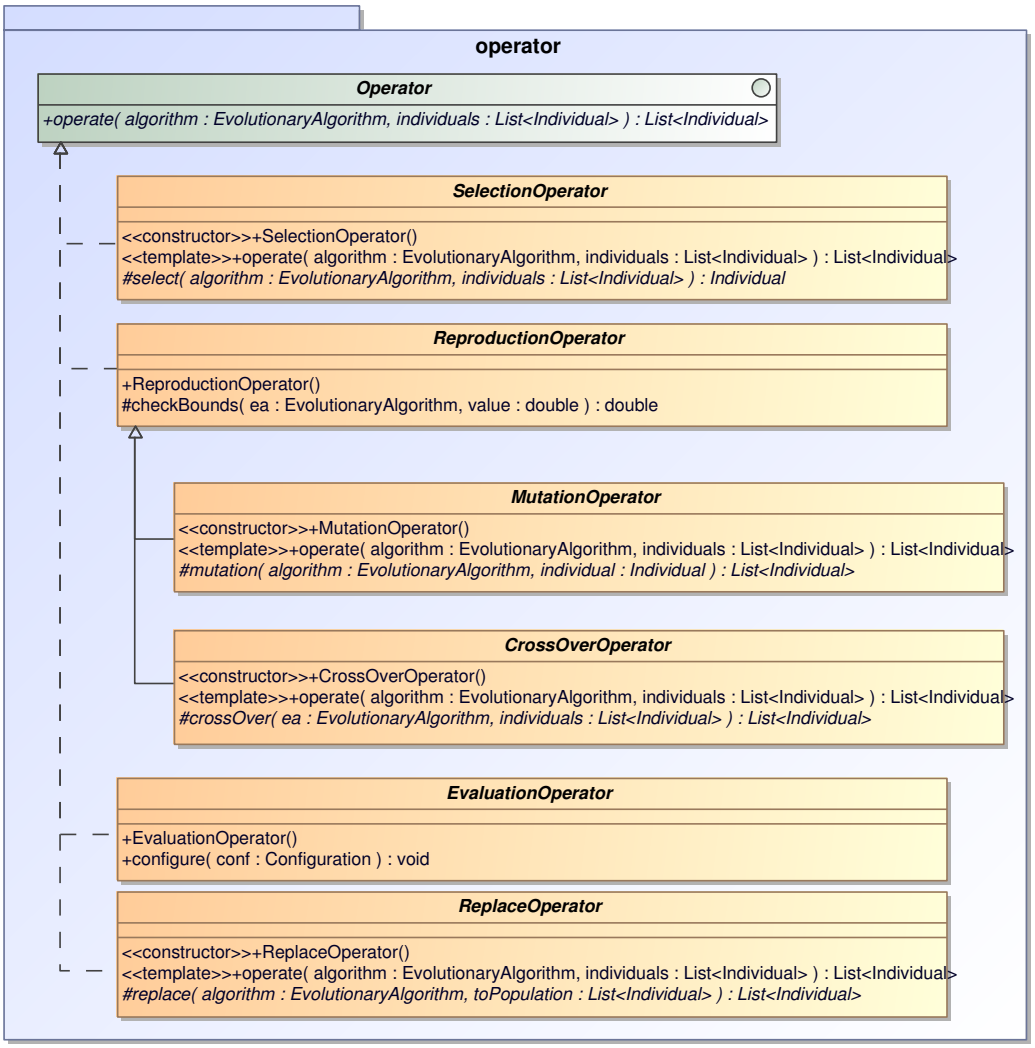


Figura A.4: Diagrama de clases del paquete *operator*.

Sistema de plug-ins Como ya fue explicado en la sección A.2, con el objetivo de conseguir un alto nivel de flexibilidad, se implementó un sistema de *plug-ins*. Principalmente se utilizan para modificar los operadores de los algoritmos, aunque se pueden utilizar en cualquier punto de la librería. Este sistema permite al usuario un alto grado de personalización sin realizar grandes esfuerzos de codificación. Por ejemplo, los parámetros de los operadores se implementan como instancias de la clase *Parameter*, esto nos permite utilizar tanto parámetros constantes como adaptativos sin necesidad de implementar un nuevo operador, simplemente configurando el operador con el tipo de parámetro deseado. En la versión actual se implementan los *plug-ins* que aparecen en el diagrama de clases de la figura A.7 y que se describen a continuación:

- **Selección de individuo (*IndividualChooser*):** el comportamiento de este *plug-in* es el de elegir un individuo en la población siguiendo un criterio dado, por ejemplo, el mejor individuo, el más cercano o un individuo aleatorio. Actualmente este *plug-in* se utiliza en la implementación del algoritmo macroevolutivo y del Differential Evolution.

- **Clasificación del NSGA2 (*NSGA2Ranking*):** este *plug-in* implementa una clase encargada de calcular la posición de un individuo en una población. Es necesario para la

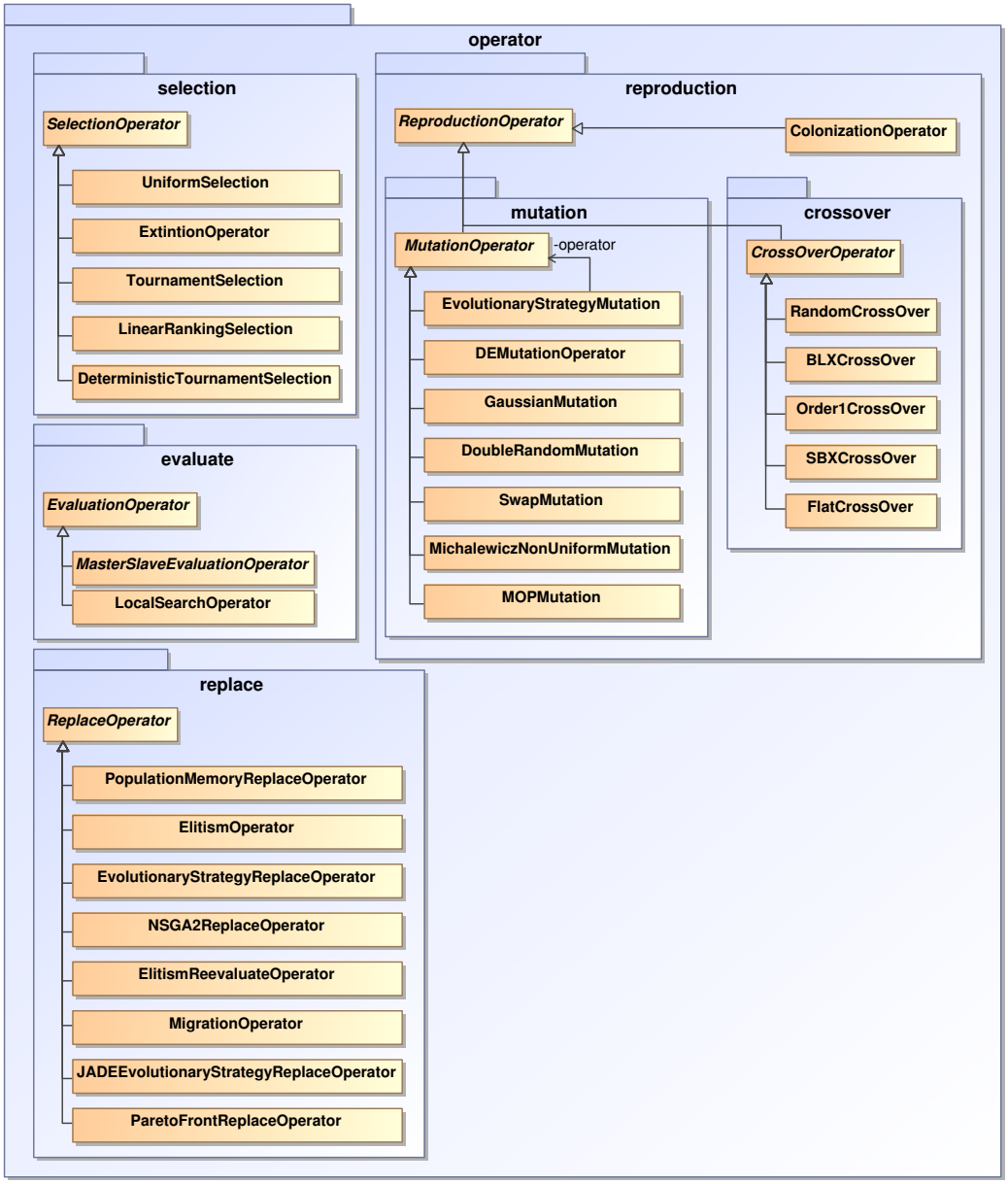


Figura A.5: Operadores implementados en la versión actual de la librería JEAF.

implementación del algoritmo NSGA2. Se ha implementado como *plug-in* para facilitar la comparación de diferentes criterios de ordenación de la población y para permitir su reutilización en otros algoritmos que puedan necesitarlo para su funcionamiento.

- **Agrupamiento (*Crowding*):** también es necesario es la implementación del algoritmo NSGA2. Calcula la distancia entre individuos de la población sirviendo como índice de agrupamiento. Existen dos implementaciones concretas que calculan la distancia basándose en distintos parámetros. Uno de ellos considera los valores de las funciones objetivo y el otro considera los valores de los parámetros del individuo.

- **Parámetros (*Parameter*):** este tipo de *plug-in* implementa diferentes formas de calcular valores de parámetros. Actualmente están implementadas cuatro sub-clases: parámetros constantes, con valores aleatorios, parámetros que decrecen de manera lineal y parámetros que decrecen de manera logarítmica. Es una herramienta muy útil que

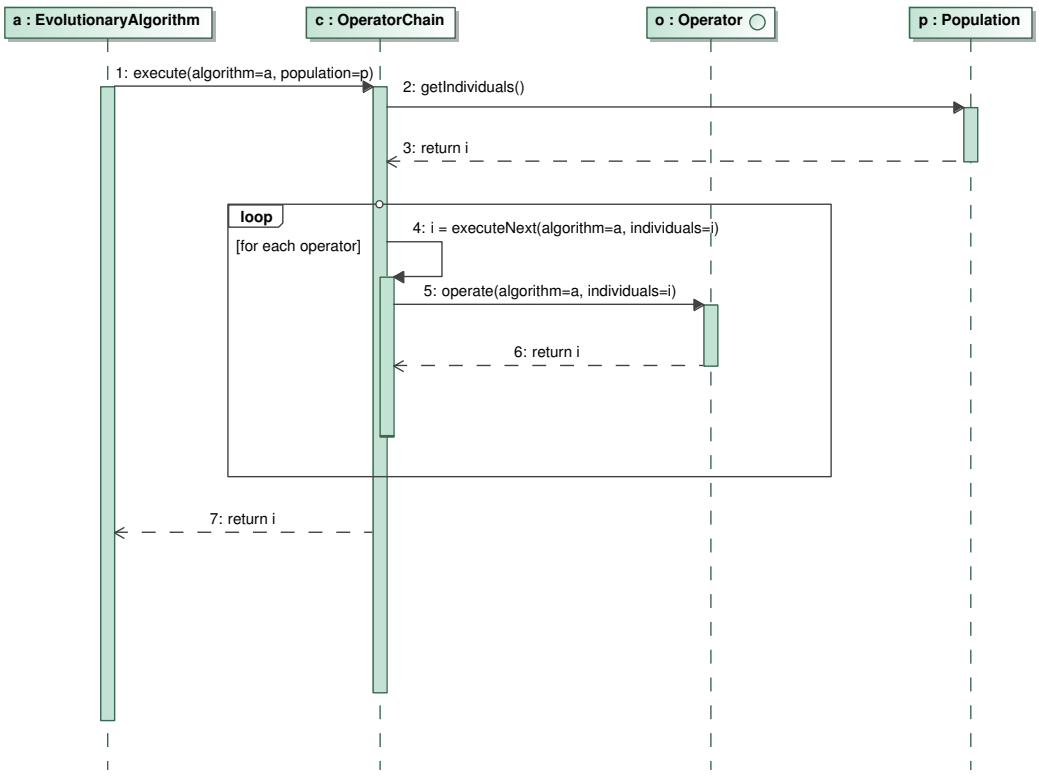


Figura A.6: En este diagrama de secuencia se describe el comportamiento de las cadenas de operadores.

permite analizar el rendimiento de los AEs dependiendo del comportamiento de los parámetros que gobiernan el funcionamiento de sus operadores.

- **Criterio de parada (*StopTestPlugin*):** los *plug-ins* que extienden esta clase son responsables de devolver el número de generaciones o de llamadas a la función de calidad que han sido ejecutadas por el algoritmo. Este *plug-in* es utilizado por los operadores cuyo comportamiento depende del número de generaciones o llamadas a la función de calidad ejecutadas, como por ejemplo el operador de mutación no uniforme de Michalewicz (clase *MichalewiczNonUniformMutation*).

Problema

Los diversos problemas susceptibles de ser resueltos por un algoritmo de optimización pueden ser clasificados, por ejemplo, según el número de funciones objetivo o el tipo de las restricciones. En la librería JEAF es posible definir cualquier combinación de las características anteriores. Según el tipo de funciones objetivo podemos distinguir:

- **Problemas mono-objetivo:** son problemas en los que solamente existe una función a optimizar, ya sea maximizar o minimizar.

- **Problemas multi-objetivo:** en este tipo de problemas existe una lista de objetivos a optimizar, generalmente no existe una solución única a todos los objetivos y el usuario deberá decidir cual es la solución que más le conviene en cada caso y que suele ser una solución compromiso entre todos los objetivos.

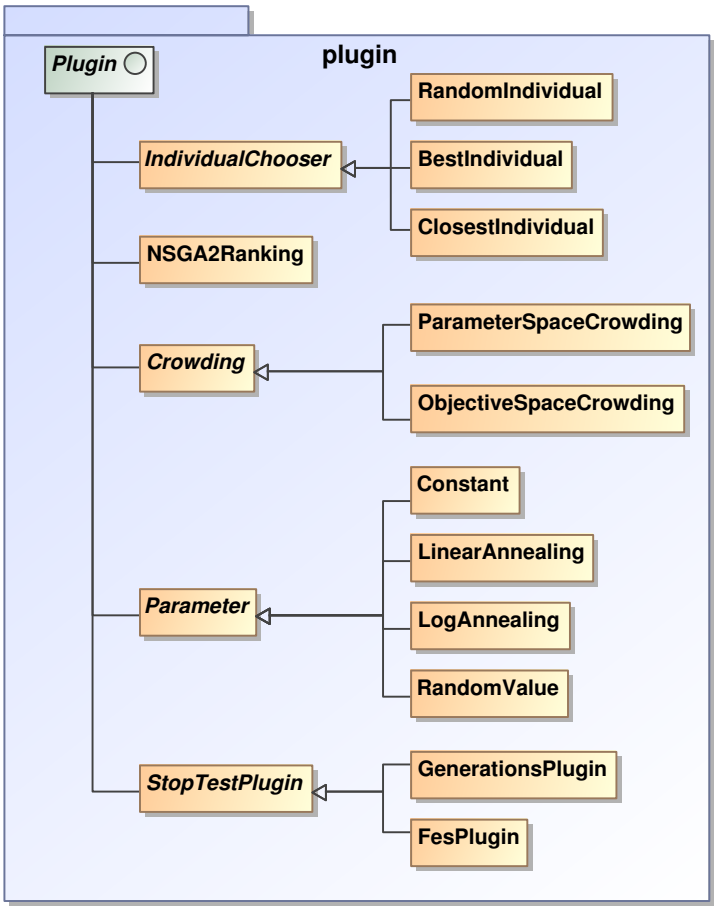


Figura A.7: Diagrama de clases del paquete *plugin*, se representan los *plug-ins* implementados en la versión actual de la librería.

Las restricciones representan limitaciones de los problemas y según el tipo de estas podemos clasificarlos como:

- **Problemas sin restricciones:** son problemas en los que solamente existen *restricciones de límite* en las variables, es decir, cada variable está definida dentro de un rango específico. Las funciones objetivo son las encargadas de decodificar los genes en el rango que le corresponde a cada uno.

- **Problemas con restricciones:** los problemas de este tipo, además de tener *restricciones de límite* en las variables, se definen mediante *funciones de restricción*. Este tipo de funciones representan relaciones complejas y no lineales entre las variables del problema y generan áreas no factibles dentro del espacio de búsqueda. Existen dos tipos de funciones de restricción: funciones de igualdad y funciones de desigualdad.

En el diagrama de clases de la figura A.8 se representa la estructura de este paquete. En JEAF es posible implementar cualquier problema combinando las características que han sido comentadas en este apartado.

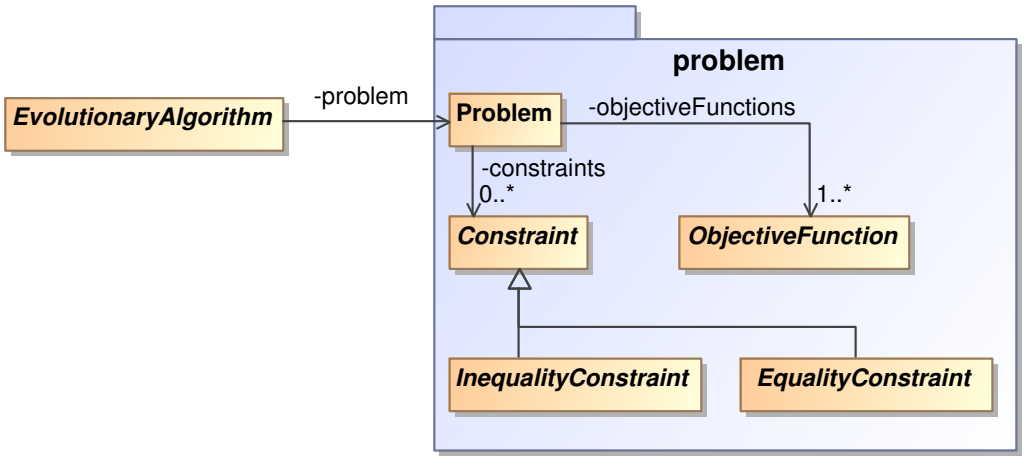


Figura A.8: Este diagrama de clases representa el paquete *problem*.

Estrategia de evaluación

Como se ha comentado en el apartado anterior, en JEAF es posible definir cualquier tipo de problema de optimización combinando una serie de características (número de funciones objetivo y tipo de restricciones). Una vez definidos, para poder resolverlos es necesario un mecanismo que adapte su comportamiento a cada tipo particular. En JEAF esto se consigue mediante las estrategias de evaluación y la elección correcta de los operadores. Durante la fase de evaluación, la lista de individuos se evalúa utilizando las funciones objetivo y las funciones de restricción, si éstas existen. El comportamiento de la estrategia de evaluación se explica en el diagrama de secuencia de la figura A.9.

En el caso de los problemas mono-objetivo y multi-objetivo, la estrategia de evaluación se encarga de evaluar a los individuos con todas las funciones objetivo definidas por el usuario y asignar a cada individuo el valor obtenido en la lista de valores objetivo. Es responsabilidad del usuario decidir que operadores utiliza en cada caso (problemas mono-objetivo o multi-objetivo). En la versión actual de JEAF existen implementaciones de los operadores necesarios para utilizar dos algoritmos multi-objetivo ampliamente conocidos: el NSGA2 [Deb et al., 2002] y el microgenético [Coello and Pulido, 2001].

En el caso de los problemas sin restricciones es responsabilidad del usuario definir los rangos límite de cada variable en las funciones objetivo, si estos límites existieran. Para lidiar con problemas con restricciones, el usuario deberá indicar qué método de manejo de restricciones (clase *ConstraintMethod*) se utilizará. La estrategia de evaluación es la encargada de dirigir el comportamiento de esta clase.

Como ya se ha mencionado en numerosas ocasiones, la fase de evaluación es la más costosa desde el punto de vista computacional. Para acelerar esta fase, se han implementado herramientas para la paralelización de la evaluación de individuos. El comportamiento de estas herramientas de paralelización está dirigido por un tipo de estrategia de evaluación distribuido. Los modelos concretos de herramientas de paralelización implementados serán explicadas en detalle en la sección A.3.4.

Además de las dos implementaciones de estrategias de evaluación que existen actualmente en la librería (*SerialEvaluationStrategy* y *DistributedEvaluationStrategy*), el usuario puede implementar cualquier tipo de estrategia de evaluación siempre y cuando siga la firma establecida en la interfaz *EvaluationStrategy*.

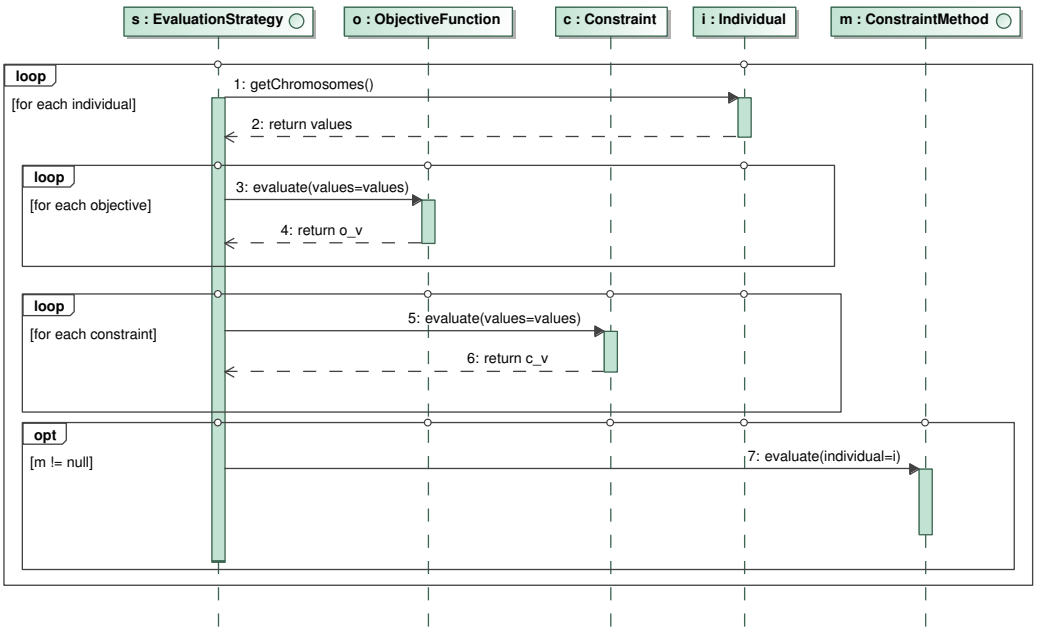


Figura A.9: Diagrama de secuencia que representa el comportamiento de las estrategias de evaluación en serie.

Criterios de parada

Para la resolución de problemas, los algoritmos de optimización iterativos, como los AEs, siguen un proceso cíclico en el cual en cada iteración (llamada generación en los AEs) se genera un nuevo conjunto de soluciones. En este tipo de algoritmo el usuario debe de indicar cuándo finaliza el proceso iterativo. En JEAF esto se realiza mediante los criterios de parada (clase *StopTest*), los cuales representan la condición para finalizar la ejecución. Como criterios de parada podemos encontrar: máximo número de generaciones, máximo número de llamadas a la función de evaluación, alcanzar un cierto valor objetivo, etc.

En esta librería, los criterios de parada pueden implementarse como criterios de parada simples o como composición de criterios. La composición de criterios ha sido implementada como una operación *or*, es decir, cuando uno de los criterios se cumple el algoritmo termina su ejecución. El diagrama de clases de este paquete puede verse en la figura A.10.

En la versión actual de JEAF existen implementaciones para los siguientes criterios de parada:

- En los criterios de parada *BitwiseConvergence* y *BestMeanConvergence*, el final del ciclo evolutivo depende del nivel de convergencia de la población. En el primero, la convergencia se mide a nivel de cromosoma como la media de bits que comparten los individuos de la población. En el segundo de los casos, la convergencia se mide a nivel fenotípico, es decir, a nivel de calidad de individuos.

- Los criterios de parada *DimensionFEsStopTest* y *MaxFEsStopTest* implementan criterios que dependen del número de llamadas a la función de evaluación. En estos dos casos, el número de iteraciones ejecutadas depende de las llamadas que se realicen a la función de evaluación. En el primero de los casos, este número de llamadas depende de la dimensión del problema, es decir, del número de genes de los cromosomas. En el segundo caso, el usuario indica un número máximo de evaluaciones.

- En el criterio *PerformanceFitnessStopTest* el usuario debe de indicar el valor de calidad que desea alcanzar en la ejecución del algoritmo. Cuando el mejor individuo obtiene ese valor de calidad, entonces finaliza la ejecución.

- Por último, en los criterios de parada *EvolveGenerationsStopTest* y *MicroGenerations-Convergence* el final del ciclo ejecutivo depende del número de generaciones o iteraciones que se hayan ejecutado.

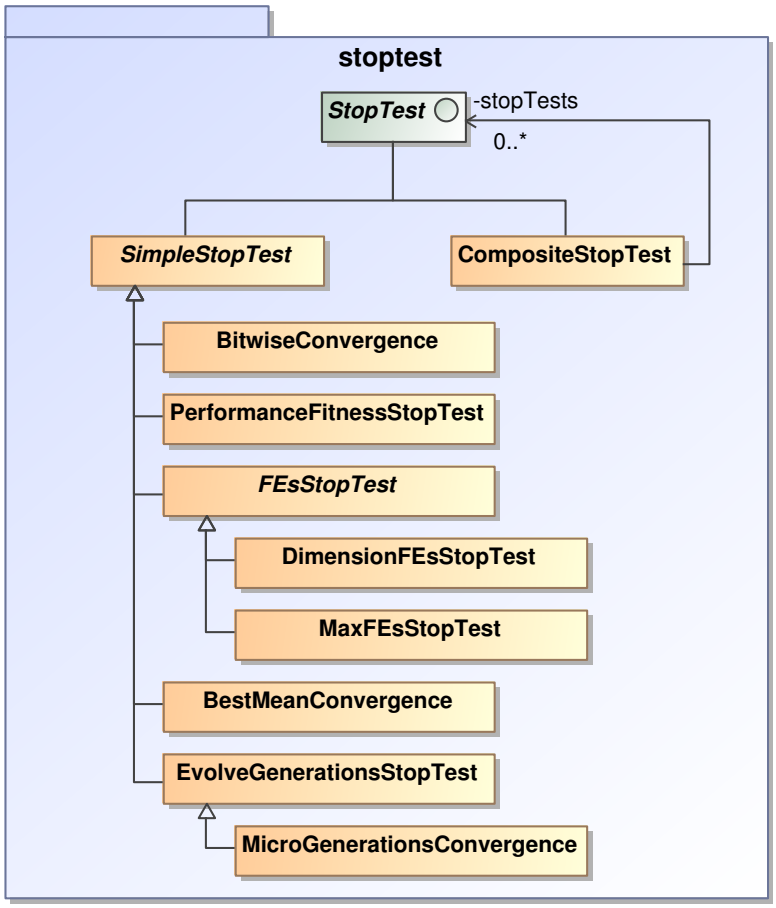


Figura A.10: Diagrama de clases del paquete *StopTest* donde aparecen los criterios de para implementados en la versión actual.

A.3.2 Módulo de funciones de prueba

El módulo de funciones de prueba ha sido implementado con el objetivo de proporcionar un conjunto de la funciones de prueba más comunes. Actualmente, los conjuntos de funciones de prueba implementados son los siguientes:

- Las funciones de prueba de la sesión especial del CEC 2005 sobre optimización de problemas de codificación real [Suganthan et al., 2005].

- Las funciones de prueba de la sesión especial del CEC 2006 sobre optimización de problemas de codificación real con restricciones [Liang et al., 2006].

- Las funciones de prueba de la sesión especial del CEC 2007 sobre problemas multi-objetivo [Huang et al., 2007].
- El conjunto de funciones de Dixon-Szegö [Dixon and Szegö, 1978].
- Otras funciones conocidas y muy utilizadas que pueden encontrarse en [Yao et al., 1999, Yao et al., 2003].

Estos conjuntos de funciones de prueba incluyen funciones con diferentes e interesantes características, como distintos tipos de modalidad, funciones separables y no separables, problemas mono-objetivo y multi-objetivo y problemas con y sin restricciones.

A.3.3 Módulo de herramientas de análisis

Además de servir como herramienta para la resolución de problemas de optimización utilizando algoritmos evolutivos, esta librería proporciona herramientas de análisis con el objetivo de facilitar el análisis de algoritmos evolutivos o de comparar el rendimiento de diferentes versiones del mismo algoritmo evolutivo.

Herramientas de análisis de algoritmos evolutivos

Con el fin de facilitar el análisis de algoritmo evolutivos han sido desarrolladas las siguientes herramientas:

- **Herramienta de análisis poblacional:** permite al usuario realizar una comparación de rendimiento del algoritmo evolutivo variando el tamaño de la población del mismo. Se ejecuta el mismo algoritmo frente al mismo problema variando el tamaño de la población.
- **Herramienta de análisis dimensional:** es muy similar a la herramienta anterior; la diferencia es que en esta herramienta el parámetro que varia y del cual se analiza la influencia sobre el rendimiento del algoritmo es la dimensionalidad del problema.
- **Herramienta de análisis de múltiples funciones:** permite definir una lista de funciones objetivo y analizar un algoritmo frente a dichas funciones.
- **Herramientas de análisis compuestos:** permite concatenar herramientas de análisis y ejecutarlas como una cadena.

Herramientas de análisis para funciones objetivo

El objetivo principal de estas herramientas es el de realizar la caracterización de algoritmos evolutivos en términos de qué tipo de funciones son más adecuadas para ser resueltas por cada algoritmo. Con este objetivo en mente, se han desarrollado dos herramientas de caracterización de funciones objetivo.

- **Herramienta de análisis de separabilidad:** esta herramienta analiza las funciones objetivo en términos de separabilidad o, en otras palabras, en términos de dependencias entre variables. La separabilidad es una característica ampliamente utilizada en el campo de los algoritmos evolutivos ya que está relacionada con la epístasis. Esta herramienta permite medir el grado de separabilidad de una función.

- **Herramienta de análisis de modalidad:** la modalidad (cuántos óptimos locales y globales existen y dónde) es otra característica importante cuando se analizan funciones para ser resueltas por algoritmos evolutivos. Con este objetivo, se ha desarrollado una herramienta que permite estimar la distribución de los óptimos en una función.

A.3.4 Paralelización

Los algoritmos evolutivos son procesos estocásticos con un alto grado de paralelización. Esta característica puede ser explotada con el fin de reducir el coste temporal de encontrar una solución a un problema o para tratar de generar mayor diversidad durante el proceso de búsqueda. Con estos dos objetivos en mente se han añadido varios componentes de paralelización a la librería que hacen que el uso de sistemas distribuidos para la ejecución de los algoritmos sea una tarea sencilla.

Estos componentes posibilitan que la ejecución de algoritmos evolutivos distribuidos se realice de manera rápida, concisa y sencilla. En la librería están implementados tres de los cuatro tipos tradicionalmente identificados [Cantú-Paz, 1995]. Estos tres métodos son: la evaluación distribuida de la población (modelo global), el modelo de islas, un modelo de grano grueso y la combinación de los modelos anteriores (modelo híbrido). Estos modelos son apropiados para arquitecturas con memoria distribuida como los clusters. Los modelos de grano fino, es decir, los modelos que requiere poblaciones superpuestas y para los cuales se recomienda el uso de una arquitectura memoria distribuida, no están implementaos en esta librería.

La clase que dirige el proceso de paralelización implementa el patrón decorador donde el objeto decorado es cualquier algoritmo evolutivo con evaluación en serie. Por lo tanto, todo algoritmo evolutivo implementado en la librería y los nuevos algoritmos desarrollados por los usuarios puede ser ejecutado de forma distribuida sin realizar ninguna codificación adicional. En la figura A.11 se muestra el diagrama de clases de las herramientas que permiten la paralelización de algoritmos evolutivos en esta librería.

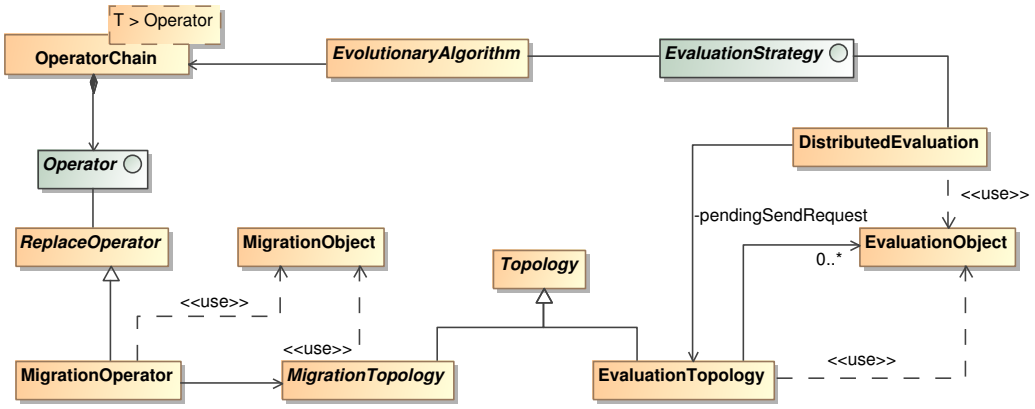


Figura A.11: Herramientas de paralelización de la librería JEAF

Las clases que permiten la paralelización de algoritmos están distribuidas en tres capas. La capa inferior es la responsable del paso de mensajes, actualmente se utiliza la librería MPJ express [Baker et al., 2006] para la implementación de esta capa. Esta librería está desarrollada completamente en JAVA e implementa la especificación de MPI versión 1.2. Por encima de esta capa, la capa intermedia permite la abstracción de las comunicaciones y hace que la

capa superior sea independiente de la librería concreta de paso de mensajes minimizando el impacto que pudiera producir el reemplazo de esta librería. La capa intermedia define varias topologías que pueden ser utilizadas directamente o extendidas para implementar nuevos modelos de conexión.

Para terminar, la capa superior tiene las clases que implementan los modelos de paralelización que se mencionaron anteriormente. Además permite que cada uno de los nodos que se utilicen pueda almacenar en disco la información de la evolución durante el proceso de búsqueda.

Actualmente existen predefinidas topologías para el modelo de islas (totalmente conectado y conexiones en anillo, en rejilla e hipercupo) y para el modelo de evaluación distribuida (topología maestro-esclavo). El modelo de islas permite tener una serie de poblaciones aisladas que se comunican mediante la migración de individuos con el fin de mejorar la diversidad global. Para dar soporte a este modelo es necesario implementar un operador de migración (que es una subclase del operador de reemplazo) que permite al usuario escoger la topología de interconexión, si la migración es síncrona o asíncrona y qué individuos son aceptados y cuales son enviados a otros nodos. Implementando la migración como un operador de reemplazo permite diseñar un algoritmo evolutivo con diferentes modelos de migración, cada uno de ellos representado por un operador.

En el modelo global, aquel que utiliza la topología maestro-esclavo como modelo de evaluación distribuido, los recursos computacionales son explotados utilizando tanto el maestro como los nodos esclavos para evaluar la población. El nodo maestro comparte la población con el resto de los nodos bajo demanda, enviando sub-conjuntos (de un tamaño prefijado) de individuos de la población. Este método permite un uso más racional del sistema distribuido en el caso de ejecutar el algoritmo evolutivo en un sistema hardware heterogéneo porque minimiza el tiempo de espera producido por nodos lentos. En la figura A.12 se muestran algunas de las posibles configuraciones para la distribución de la población durante la ejecución de un algoritmo evolutivo.

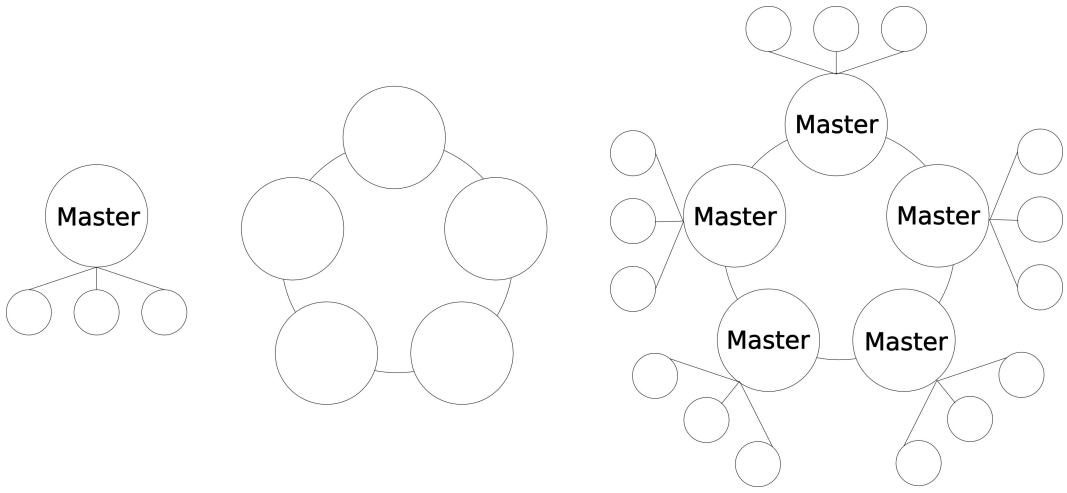


Figura A.12: Algunos de los modelos de paralelización implementados en JEAF. De izquierda a derecha: maestro-esclavo, islas con topología en rejilla e islas con una topología en islas combinado con un modelo maestro-esclavo.

En este apéndice ha sido explicado el diseño de la librería JEAF utilizada para el desarrollo de esta tesis y que se encuentra disponible en www.gii.udc.es/jeaf.

Apéndice B

Funciones objetivo utilizadas en esta tesis

1. **Función Ackley's**

$$f(x) = -20.0 \exp\left(-0.2 * \sqrt{\frac{1.0}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1.0}{n}\sum_{i=1}^{n} cos(2.0\pi x_i)\right) + 20.0 - e \tag{B.1}$$

donde los límites de las variables son $-30.0 \leq x_i \leq 30.0$. El óptimo global de la función está situado en el punto $x^* = (0, \ldots, 0)$ y su valor de calidad es $f(x^*) = 0$.

2. **Función Aluffi-Pentini's**

$$f(x) = 0.25x_1^4 - 0.5x_1^2 + 0.1x_1 + 0.5x_2^2 \tag{B.2}$$

donde los límites de las variables son $-10.0 \leq x_i \leq 10.0$. El óptimo global de la función está situado en el punto $x^* = (-1.0465, 0)$ y su valor de calidad es $f(x^*) \approx -0.3523$.

3. **Función Axis Parallel Hyperellipsoid**

$$f(x) = \sum_{i=1}^{n} ix^2 \tag{B.3}$$

donde los límites de las variables son $-100.0 \leq x_i \leq 100.0$. El óptimo global de la función está situado en el punto $x^* = (0, \ldots, 0)$ y su valor de calidad es $f(x^*) = 0$.

4. **Función Beale**

$$\begin{aligned} f(x) = &(1.5 - x_1(1.0 - x_2))^2 \\ &+ (2.25 - x_1(1 - x_2^2))^2 \\ &+ (2.625 - x_1(1.0 - x_2^3))^2 \end{aligned} \tag{B.4}$$

donde los límites de las variables son $-4.5 \leq x_i \leq 4.5$. El óptimo global de la función está situado en el punto $x^* = (3.0, 0.5)$ y su valor de calidad es $f(x^*) = 0$.

5. **Función Becker and Lago**

$$f(x) = (x_1 - 5.0)^2 + (x_2 - 5.0)^2 \tag{B.5}$$

donde los límites de las variables son $-10.0 \leq x_i \leq 10.0$. El óptimo global de la función está situado en el punto $x^* = (\pm 5, \pm 5)$ y su valor de calidad es $f(x^*) = 0$.

6. **Función Bohachevsky 1**

$$f(x) = x_1^2 + 2x_2^2 - 0.3cos(3\pi x_1) - 0.4cos(4\pi x_2) + 0.7 \tag{B.6}$$

donde los límites de las variables son $-50.0 \leq x_i \leq 50.0$. El óptimo global de la función está situado en el punto $x^* = (0, 0)$ y su valor de calidad es $f(x^*) = 0$.

7. **Función Bohachevsky 2**

$$f(x) = x_1^2 + 2x_2^2 - 0.3cos(3\pi x_1)cos(4\pi x_2) + 0.3 \tag{B.7}$$

donde los límites de las variables son $-50.0 \leq x_i \leq 50.0$. El óptimo global de la función está situado en el punto $x^* = (0, 0)$ y su valor de calidad es $f(x^*) = 0$.

8. **Función Colville**

$$\begin{aligned} f(x) = {} & 100(x_1^2 - x_2))^2 + (x_1 - 1.0)^2 \\ & + (x_3 - 1.0)^2 + 90.0 * (x_3^2 - x_4)^2 \\ & + 10.1 * ((x_2 - 1.0)^2 + (x_4 - 1.0)^2) \\ & + 19.8 * (x_2^-1.0) * (x_4 - 1.0) \end{aligned} \tag{B.8}$$

donde los límites de las variables son $-10.0 \leq x_i \leq 10.0$. El óptimo global de la función está situado en el punto $x^* = (1.0, \ldots, 1.0)$ y su valor de calidad es $f(x^*) = 0$.

9. **Función Cosine Mixture**

$$f(x) = 0.1 \sum_{i=1}^{n} cos(5.0\pi x_i) - \sum_{i=1}^{n} x_i^2 \tag{B.9}$$

donde los límites de las variables son $-1.0 \leq x_i \leq 1.0$. El óptimo global de la función está situado en el punto $x^* = (0, \ldots, 0)$ y su valor de calidad es $f(x^*) = 1/n$.

10. **Función Dekkers and Aarts**

$$f(x) = 10^4 x_1^2 + x_2^2 - (x_1^2 + x_2^2)^2 + \frac{(x_1^2 + x_2^2)^4}{10^4} \tag{B.10}$$

donde los límites de las variables son $-20.0 \leq x_i \leq 20.0$. El óptimo global de la función está situado en el punto $x^* = (0, 15)$ and $(0, -15)$ y su valor de calidad es $f(x^*) \approx -24777$.

i	c_i	a_{ij}			p_{ij}		
		j = 0	1	2	j = 0	1	2
0	1.0	3.0	10.0	30.0	0.36890	0.11700	0.26730
1	1.2	0.1	10.0	35.0	0.46990	0.43870	0.74700
2	3.0	3.0	10.0	30.0	0.10910	0.87320	0.55470
3	3.2	0.1	10.0	30.0	0.03815	0.57743	0.88280

Tabla B.1: Parámetros de la función Hartman 3

11. **Función Easom**

$$\begin{aligned} f(x) =& cos(x_1)cos(x_2) \\ & \exp(-(x_1-\pi)^2-(x_2-\pi)^2) \end{aligned} \tag{B.11}$$

donde los límites de las variables son $-100.0 \leq x_i \leq 100.0$. El óptimo global de la función está situado en el punto $x^* = (\pi, \pi)$ y su valor de calidad es $f(x^*) = -1.0$.

12. **Función Griewank**

$$f(x) = 1.0 + \frac{1.0}{4000.0}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} cos\frac{x_i}{\sqrt{i}} \tag{B.12}$$

donde los límites de las variables son $-600.0 \leq x_i \leq 600.0$. El óptimo global de la función está situado en el punto $x^* = (0, \ldots, 0)$ y su valor de calidad es $f(x^*) = 0$.

13. **Función Goldstein Price**

$$\begin{aligned} f(x) =& (1.0 + (x_1 + x_2 + 1.0)^2(19.0 - 14.0x_1 + \\ & 3.0x_1 - 14.0x_2 + 6.0x_1x_2 + 3.0x_2^2)) \\ & (30.0 + (2.0x_1 - 3.0x_2)^2(18.0 - 32.0x_1 + \\ & 12.0x_1^2 + 48.0x_2 - 36.0x_1x_2 + 27.0x_2^2)) \end{aligned} \tag{B.13}$$

donde los límites de las variables son $-2.0 \leq x_i \leq 2.0$. El óptimo global de la función está situado en el punto $x^* = (0, -1)$ y su valor de calidad es $f(x^*) = 3$.

14. **Función Hartman 3**

$$f(x) = -\sum_{i=1}^{3} c_i \exp\left[\sum_{i=1}^{3} a_{ij}(x_i - p_{ij})^2\right] \tag{B.14}$$

donde los límites de las variables son $0 \leq x_i \leq 1$. Esta función presenta cuatro mínimos locales, $x_{local} \approx (p_{i0}, p_{i1}, p_{i2})$ con $f(x_{local}) \approx -c_i$. El óptimo global es $x^* \approx (0.114614, 0.555649, 0.852547)$ y su valor de calidad es $f(x^*) \approx -3.862782$.

15. **Función Hartman 6**

$$f(x) = -\sum_{i=1}^{6} c_i \exp\left[\sum_{i=1}^{6} a_{ij}(x_i - p_{ij})^2\right] \tag{B.15}$$

donde los límites de las variables son $0 \leq x_i \leq 1$ B.2. Esta función presenta cuatro mínimos locales, $x_{local} \approx (p_{i0}, \ldots, p_{i5})$ con $f(x_{local}) \approx -c_i$. El óptimo global es $x^* \approx$ (0.201690, 0.150011, 0.476874, 0.275332, 0.311652, 0.657301) y su valor de calidad es $f(x^*) \approx -3.322368$.

i	c_i	a_{ij}					
		j = 0	1	2	3	4	5
0	1.0	10.00	3.00	17.00	3.50	17.00	8.00
1	1.2	0.05	10.00	17.00	0.10	8.00	14.00
2	3.0	3.00	3.50	1.70	10.00	17.00	8.00
3	3.2	17.00	8.00	0.05	10.00	0.10	14.00

Tabla B.2: Parámetros de la función Hartman 6

i	c_i	p_{ij}			.		
		j = 0	1	2	3	4	5
0	1.0	0.1312	0.1696	0.5569	0.0124	0.8283	0.5886
1	1.2	0.2329	0.4135	0.8307	0.3736	0.1004	0.9991
2	3.0	0.2348	0.1451	0.3522	0.2883	0.3047	0.6650
3	3.2	0.4047	0.8828	0.8732	0.5743	0.1091	0.0381

Tabla B.3: Parámetros de la función Hartman 6

16. **Función Kowalik's**

$$min_x f(x) = \sum_{i=0}^{10} \left(a_i - \frac{x_1(b_i^2 + b_i x_2)}{b_i^2 + b_i x_3 + x_4}\right)^2 \tag{B.16}$$

donde los límites de las variables son $-5.0 \leq x_i \leq 5.0$. El óptimo global de la función está situado en el punto $x^* \approx (0.192, 0.190, 0.123, 0.135)$ y su valor de calidad es $f(x^*) \approx 3.0748 \times 10^{-4}$.

17. **Función Levy**

$$f(x) = sin^2(\pi y_1) + (y_n - 1.0)^2 + \sum_{i=1}^{n-1}\left[(y_i - 1.0)^2(1.0 + 10.0 sin^2(\pi y_{i+1}))\right] \tag{B.17}$$

i	0	1	2	3	4	5
a_i	0.1957	0.1947	0.1735	0.1600	0.0844	0.0627
b_i	0.2500	0.5000	1.0000	2.0000	4.0000	6.0000
i	6	7	8	9	10	
a_i	0.0456	0.0342	0.0323	0.0235	0.0246	.
b_i	8.0000	10.0000	12.0000	14.0000	16.0000	.

Tabla B.4: Parámetros de la función Kowalik's

donde $y_i = 1.0 + \frac{x_i - 1.0}{4.0}$. Los límites de las variables son $-10.0 \leq x_i \leq 10.0$ El óptimo global es $x^* = (1, \ldots, 1)$ y su valor de calidad es $f(x^*) = 0$.

18. **Función Matyas**

$$f(x) = 0.26(x_1^2 + x_2^2) - 0.48x_1x_2 \tag{B.18}$$

donde los límites de las variables son $-10.0 \leq x_i \leq 10.0$. El óptimo global de la función está situado en el punto $x^* = (0, \ldots, 0)$ y su valor de calidad es $f(x^*) = 0$.

19. **Función Penalized 1**

$$\begin{aligned} f(x) = \frac{\pi}{n}\{10sin^2(\pi y_1) + \sum_{i=2}^{n-1}(y_i - 1)^2 \\ \cdot \left[1 + 10sin^2(\pi y_i)\right] + (y_n - 1)^2\} \\ + \sum_{i=1}^{n} u(x_i, 10, 100, 4) \end{aligned} \tag{B.19}$$

donde los límites de las variables son $-50.0 \leq x_i \leq 50.0$. El óptimo global de la función está situado en el punto $x^* = (-1, \ldots, -1)$ y su valor de calidad es $f(x^*) = 0$.

20. **Función Penalized 2**

$$\begin{aligned} f(x) = 0.1\{sin^2(\pi 3x_1) + \sum_{i=2}^{n-1}(x_i - 1)^2 \\ \cdot \left[1 + sin^2(\pi 3x_i)\right] + (x_n - 1)^2\left[1 + 2sin^2(2\pi x_n)\right]\} \\ + \sum_{i=1}^{n} u(x_i, 5, 100, 4) \end{aligned} \tag{B.20}$$

donde los límites de las variables son $-50.0 \leq x_i \leq 50.0$. El óptimo global de la función está situado en el punto $x^* = (1, \ldots, 1)$ y su valor de calidad es $f(x^*) = 0$.

En las funciones Penalized 1 y Penalized 2 functions, los valores de la función u y la variable y se calculan según las siguientes funciones:

$$u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m & x_i > a \\ 0 & -a \leq x_i \leq a \\ k(-x_i - a)^m & x_i < a \end{cases}$$

$$y_i = 1 + \frac{1}{4}(x_i + 1)$$

21. **Función Perm**

$$f(x) = \sum_{k=1}^{n} \left[\sum_{i=1}^{n} (i + \beta)(x_i^k - (1/i)^k) \right] \tag{B.21}$$

donde $\beta = 0.5$. Los límites de las variables son $-n \leq x_i \leq n$. El óptimo global de la función está situado en el punto $x^* = (1, 1/2, \cdots, 1/n)$ y su valor de calidad es $f(x^*) = 0$.

22. **Función Rastrigin**

$$f(x) = \sum_{i=1}^{n} (x_i^2 - 10.0cos(2.0\pi x_i + 10.0)) \tag{B.22}$$

donde los límites de las variables son $-5.12 \leq x_i \leq 5.12$. El óptimo global de la función está situado en el punto $x^* = (0, \ldots, 0)$ y su valor de calidad es $f(x^*) = 0$.

23. **Función Rosenbrock**

$$f(x) = \sum_{i=1}^{n-1} \left[100.0(x_{i+1} - x_i^2)^2 + (x_i - 1.0)^2 \right] \tag{B.23}$$

donde los límites de las variables son $-30.0 \leq x_i \leq 30.0$. El óptimo global de la función está situado en el punto $x^* = (0, \ldots, 0)$ y su valor de calidad es $f(x^*) = 0$.

24. **Función Schwefel 1.2**

$$f(x) = \sum_{i=1}^{n} \left(\sum_{j=1}^{n} x_j \right)^2 \tag{B.24}$$

donde los límites de las variables son $-100.0 \leq x_i \leq 100.0$. El óptimo global de la función está situado en el punto $x^* = (0, \ldots, 0)$ y su valor de calidad es $f(x^*) = 0$.

25. **Función Schwefel 2.21**

$$f(x) = \max_{i=1}^{n}(x_i) \tag{B.25}$$

donde los límites de las variables son $-100.0 \leq x_i \leq 100.0$. El óptimo global de la función está situado en el punto $x^* = (0, \ldots, 0)$ y su valor de calidad es $f(x^*) = 0$.

26. **Función Schwefel 2.22**

$$f(x) = \sum_{i=1}^{n} |x_i| + \prod_{i=1}^{n} |x_i| \tag{B.26}$$

donde los límites de las variables son $-10.0 \leq x_i \leq 10.0$. El óptimo global de la función está situado en el punto $x^* = (0, \ldots, 0)$ y su valor de calidad es $f(x^*) = 0$.

27. **Función Schwefel**

$$f(x) = -\sum_{i=1}^{n} x_i sin(\sqrt{|x_i|}) \tag{B.27}$$

donde los límites de las variables son $-500.0 \leq x_i \leq 500.0$. El óptimo global de la función está situado en el punto $x^* = (420.97, \ldots, 420.97)$ y su valor de calidad es $f(x^*) \approx -418.9829n$.

28. **Función Shekel Family**

$$f(x) = -\sum_{i=1} m[(x_i - a_i)(x_i - a_i)^T + c_i]^{-1} \tag{B.28}$$

with $m = 5, 7, 10$, donde los límites de las variables son $0 \leq x_i \leq 10$. Esta función presenta cinco, siete y diez óptimos locales, $x_{(local\_minima)} \approx a_i$, y su valor de calidad es $f(x_{local\_minima}) \approx 1/c_i$, para $0 \leq i < m$.

29. **Función Shekel's Foxholes**

$$f(x) = \left[\frac{1}{500} + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2} (x_i - a_{ij})^6} \right]^{-1} \tag{B.29}$$

donde los límites de las variables son $-65.536 \leq x_i \leq 65.536$. El óptimo global de la función está situado en el punto $x^* = (-32, -32)$ y su valor de calidad es $f(x^*) \approx 1$.

30. **Función Six Hump Camel Back**

$$\begin{aligned} f(x) =& 4.0x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 \\ &+ x_1x_2 - 4.0x_2^2 + 4.0x_2^4 \end{aligned} \tag{B.30}$$

donde los límites de las variables son $-5.0 \leq x_i \leq 5.0$. El óptimo global de la función está situado en el punto $x^* = (0.08983, -0.7126), (-0.08983, 0.7126)$ y su valor de calidad es $f(x^*) \approx -1.0316$.

31. **Función Sphere Model**

$$f(x) = \sum_{i=1}^{n} x_i^2 \tag{B.31}$$

donde los límites de las variables son $-100.0 \leq x_i \leq 100.0$. El óptimo global de la función está situado en el punto $x^* = (0, \ldots, 0)$ y su valor de calidad es $f(x^*) = 0$.

32. **Función Step**

$$f(x) = \sum_{i=1}^{n} (\mid x_i + 0.5 \mid)^2 \tag{B.32}$$

donde los límites de las variables son $-100.0 \leq x_i \leq 100.0$. El óptimo global de la función está situado en el punto $x^* = (0, \ldots, 0)$ y su valor de calidad es $f(x^*) = 0$.

33. **Función Sum of Different Power**

$$f(x) = \sum_{i=1}^{n} x_i^i \tag{B.33}$$

donde los límites de las variables son $-1.0 \leq x_i \leq 1.0$. El óptimo global de la función está situado en el punto $x^* = (0.0, \ldots, 0.0)$ y su valor de calidad es $f(x^*) = 0$.

34. **Función Zakharov**

$$f(x) = \sum_{i=1}^{n} x_i^2 + \left(\sum_{i=1}^{n} 0.5 i x_i \right)^2 + \left(\sum_{i=1}^{n} 0.5 i x_i \right)^4 \tag{B.34}$$

donde los límites de las variables son $-5.0 \leq x_i \leq 10.0$. El óptimo global de la función está situado en el punto $x^* = (0.0, \ldots, 0.0)$ y su valor de calidad es $f(x^*) = 0$.

Referencias

[JDE, 1999] (1999). JDEAL. URL: http://www.laseeb.org/sw/JDEAL/home.html.

[TEA, 2003] (2003). TEA. URL: http://sfbci.unidortmund.de/index.php?option=com_ content& task=view &id=28& Itemid=160.

[jMe, 2006] (2006). jMetal. URL: http://jmetal.sourceforge.net/.

[ssL, 2007] (2007). Special Session on Linkage in Evolutionary Computation (LEC 2007).

[GAU, 2009] (2009). GAUL. URL: http://gaul.sourceforge.net/.

[BEA, 2009] (2009). Open BEAGLE. URL: http://beagle.gel.ulaval.ca/.

[PGA, 2009] (2009). PGAPack. URL: ftp://info.mcs.anl.gov/pub/pgapack.

[ssL, 2009] (2009). Special Session on Linkage in Evolutionary Computation (LEC 2009).

[ECJ, 2010] (2010). ECJ. URL: http://cs.gmu.edu/ clab/projects/ecj/.

[Ope, 2010] (2010). OpenOpal. URL: http://openopal.origo.ethz.ch/.

[Abbass, 2002] Abbass, H. A. (2002). An Evolutionary Artificial Neural Networks Approach for Breast Cancer Diagnosis. *Artificial Intelligence in Medicine*, 25:265--281.

[Abudhahir and Baskar, 2008] Abudhahir, A. and Baskar, S. (2008). Evolutionary optimised nonlinear function for linearisation of constant temperature anemometer. *Science, Measurement & Technology, IET*, 2(4):208--218.

[Ackley, 1987] Ackley, D. (1987). *A connectionist machine for genetic hillclimbing*. Kluwer Academic Publishers, Norwell, MA, USA.

[Aguilar and Colmenares, 1998] Aguilar, J. and Colmenares, A. (1998). Resolution of pattern recognition problems using a hybrid genetic/random neural network learning algorithm. *Pattern Analysis & Applications*, 1(1):52--61.

[Albrecht et al., 2008] Albrecht, A., Lane, P., and Steinhöfel, K. (2008). Combinatorial landscape analysis for k-SAT instances. In *IEEE Congress on Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence)*, pages 2498--2504.

[Albrecht et al., 2010] Albrecht, A., Lane, P., and Steinhöfel, K. (2010). Analysis of Local Search Landscapes for k-SAT Instances. *Mathematics in Computer Science*, 3(4):465--488.

[Ali et al., 2005] Ali, M., Khompatraporn, C., and Zabinsky, Z. (2005). A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems. *Journal of Global Optimization*, 31(4):635--672.

[Altenberg, 1994] Altenberg, L. (1994). Evolving Better Representations through Selective Genome Growth. In *In Proceedings of the IEEE World Congress on Computational Intelligence*, pages 182--187. IEEE.

[Altenberg, 1997a] Altenberg, L. (1997a). Fitness distance correlation analysis: An instructive counterexample. In *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 57--64.

[Altenberg, 1997b] Altenberg, L. (1997b). NK Fitness Landscapes. *Handbook of Evolutionary Computation*, 2:2.7:5--2.7:10.

[Armananzas et al., 2008] Armananzas, R., Inza, I., Santana, R., Saeys, Y., Flores, J., Lozano, J., Peer, Y., Blanco, R., Robles, V., Bielza, C., and Larrañaga, P. (2008). A review of estimation of distribution algorithms in bioinformatics. *BioData Mining*, 1(1):6.

[Auger and Hansen, 2005] Auger, A. and Hansen, N. (2005). A Restart CMA Evolution Strategy with Increasing Population Size. In *In Proceedings of the 2005 IEEE Congress on Evolutionary Computation, 2005.*, volume 2, pages 1769--1776.

[Babu and Sastry, 1999] Babu, B. and Sastry, K. (1999). Estimation of heat transfer parameters in a trickle-bed reactor using differential evolution and orthogonal collocation. *Computers and Chemical Engineering*, 23:327--339.

[Bäck, 1994] Bäck, T. (1994). Selective pressure in evolutionary algorithms: A characterization of selection mechanisms. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, volume 1, pages 57--62.

[Bäck, 1996] Bäck, T. (1996). *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press, Oxford, UK.

[Bäck et al., 1997] Bäck, T., Fogel, D. B., and Michalewicz, Z., editors (1997). *Handbook of Evolutionary Computation*. IOP Publishing Ltd., Bristol, UK, UK.

[Bäck and Schwefel, 1993] Bäck, T. and Schwefel, H.-P. (1993). An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation*, 1(1):1--23.

[Baker et al., 2006] Baker, M., Carpenter, B., and Shafi, A. (2006). MPJ Express: Towards Thread Safe Java HPC. In *Proceedings of 2006 IEEE International Conference on Cluster Computing (Cluster 2006)*, Barcelona, Spain.

[Baluja, 1994] Baluja, S. (1994). Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Technical Report CMU-CS-94-163, Carnegie Mellon University, Pittsburgh, PA.

[Baluja and Davies, 1997] Baluja, S. and Davies, S. (1997). Using optimal dependency-trees for combinatorial optimization: Learning the structure of the search space. In *Proceedings of the 14th International Conference on Machine Learning*, pages 30--38.

[Bartz-Beielstein, 2010] Bartz-Beielstein, T. (2010). SPOT: An R Package For Automatic and Interactive Tuning of Optimization Algorithms by Sequential Parameter Optimization.

[Bartz-Beielstein and Preuss, 2007] Bartz-Beielstein, T. and Preuss, M. (2007). Experimental research in Evolutionary Computation. In *GECCO '07: Proceedings of the 2007 GECCO Conference Companion on Genetic and Evolutionary Computation*, pages 3001--3020, New York, NY, USA. ACM.

[Bartz-Beielstein and Preuss, 2010] Bartz-Beielstein, T. and Preuss, M. (2010). Tuning and Experimental Analysis in EC: What We Still Have Wrong. In *GECCO '2010: Proceedings of the 12th Annual Conference Companion on Genetic and Evolutionary Computation Conference*, New York, NY, USA. ACM.

[Becerra et al., 2005a] Becerra, J., Bellas, F., Santos, J., and Duro, R. (2005a). Complex Behaviours Through Modulation in Autonomous Robot Control. *Computational Intelligence and Bioinspired Systems*, pages 717--724.

[Becerra and Reyes, 2005] Becerra, J. A. and Reyes, J. S. (2005). Neural Clustering Analysis of Macroevolutionary and Genetic Algorithms in the Evolution of Robot Controllers. In *Proceedings of the First International Work-Conference on the Interplay Between Natural and Artificial Computation, IWINAC 2005*, pages 415--424.

[Becerra et al., 2003] Becerra, J. A., Santos, J., and Duro, R. J. (2003). Multimodule Artificial Neural Network Architectures for Autonomous Robot Control Through Behavior Modulation. In *IWANN '03: Proceedings of the 7th International Work-Conference on Artificial and Natural Neural Networks*, pages 169--176, Berlin, Heidelberg. Springer-Verlag.

[Becerra et al., 2005b] Becerra, J. A., Santos, J., and Duro, R. J. (2005b). *Information Processing with Evolutionary Algorithms From Industrial Applications to Academic Speculations*, chapter Robot Controller Evolution with Macroevolutionary Algorithms. Springer-Verlag.

[Bengoetxea et al., 2001] Bengoetxea, E., Larrañaga, P., Bloch, I., and Perchant, A. (2001). Estimation of Distribution Algorithms: A New Evolutionary Computation Approach for Graph Matching Problems. *Lecture Notes in Computer Science: Energy Minimization Methods in Computer Vision and Pattern Recognition*, 2134:454--469.

[Bentley, 1999] Bentley, P. (1999). *Evolutionary design by computers*. Morgan Kaufmann.

[Bentley and Kumar, 1999] Bentley, P. and Kumar, S. (1999). Three ways to grow designs: A comparison of embryogenies for an evolutionary design problem. In *Genetic and Evolutionary Computation Conference*, pages 35--43.

[Binder and Young, 1986] Binder, K. and Young, A. (1986). Spin glasses: Experimental facts, theoretical concepts, and open questions. *Reviews of Modern physics*, 58(4):801--976.

[Birattari et al., 2002] Birattari, M., Stützle, T., Paquete, L., and Varrentrapp, K. (2002). A Racing Algorithm for Configuring Metaheuristics. In *GECCO '02: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 11--18, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Blanco and Lozano, 2001] Blanco, R. and Lozano, J. (2001). *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*, chapter Empirical comparison of Estimation of Distribution Algorithms in combinatorial optimization. Kluwer Academic Publishers.

[Borenstein and Poli, 2005] Borenstein, Y. and Poli, R. (2005). Information landscapes and problem hardness. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 1425--1431, New York, NY, USA. ACM.

[Bornholdt, 1997] Bornholdt, S. (1997). *Foundations of Genetic Algorithms*, volume 4, chapter Probing Genetic Algorithm Performance of Fitness Landscapes, pages 141--154. Morgan Kaufmann.

[Branke, 2001] Branke, J. (2001). *Evolutionary Optimization in Dynamic Environments*. Genetic Algorithms and Evolutionary Computation. Kluwer Academic Publishers, Norwell, MA, USA.

[Cantú-Paz, 1995] Cantú-Paz, E. (1995). A Summary of Research on Parallel Genetic Algorithms. Technical Report IlliGAL Report 95007, University of Illinois at Urbana-Champaign.

[Cantú-Paz, 2002] Cantú-Paz, E. (2002). Feature Subset Selection By Estimation Of Distribution Algorithms. In *GECCO '02: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 303--310, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Chan and Dill, 1991] Chan, H. and Dill, K. (1991). Sequence Space Soup of Proteins and Copolymers. *Journal of Chemical Physics*, 95:3775--3787.

[Chan et al., 2003] Chan, K., Aydin, M., and Fogarty, T. (2003). An Epistasis Measure Based on the Analysis of Variance for the Real-coded Representation in Genetic Algorithms. In *CEC 2003: the 2003 Congress on Evolutionary Computation:[proceedings]: Canberra, Australia, 8-12 December*, page 297. IEEE.

[Chang and Chang, 1998] Chang, T. and Chang, H. (1998). Application of differential evolution to passive shunt harmonicfilter planning. In *8th International Conference on Harmonics And Quality of Power, 1998. Proceedings*, volume 1.

[Chen et al., 2002] Chen, C.-W., Chen, D.-Z., and Cao, G.-Z. (2002). An improved differential evolution algorithm in training and encoding prior knowledge into feedforward networks with application in chemistry. *Chemometrics and Intelligent Laboratory Systems*, 64(1):27 -- 43.

[Chu and Beasley, 1997] Chu, P. C. and Beasley, J. E. (1997). A genetic algorithm for the generalised assignment problem. *Computers & Operations Research*, 24(1):17--23.

[Coello and Pulido, 2001] Coello, C. A. C. and Pulido, G. T. (2001). A Micro-Genetic Algorithm for Multiobjective Optimization. In *EMO '01: Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization*, pages 126--140, London, UK. Springer-Verlag.

[Coello Coello, 1999] Coello Coello, C. (1999). A comprehensive survey of evolutionary-based multiobjective optimization techniques. *Knowledge and Information Systems*, 1(3):129--156.

[Coello Coello, 2002] Coello Coello, C. (2002). Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. *Computer methods in applied mechanics and engineering*, 191(11-12):1245--1287.

[Coll et al., 1999] Coll, P. E., Durán, G. A., and Moscato, P. (1999). On worst-case and comparative analysis as design principles for efficient recombination operators: A graph coloring case study. In *New Ideas in Optimization*, pages 279--294. McGraw-Hill.

[Costa et al., 1995] Costa, D., Hertz, A., and Dubuis, C. (1995). Embedding a sequential procedure within an evolutionary algorithm for coloring problems in graphs. *Journal of Heuristics*, 1(1):105--128.

[Cotta et al., 2001] Cotta, C., Alba, E., Sagarna, R., and Larrañaga, P. (2001). *Estimation of Distribution Algorithms: A new tool for Evolutionary Computation*, chapter Adjusting Weights in Artificial Neural Networks using Evolutionary Algorithms, pages 357--373. Genetic Algorithms and Evolutionary Computation. Kluwer Academic Publishers.

[Cotta and Troya, 2000] Cotta, C. and Troya, J. (2000). Using a Hybrid Evolutionary-A* Approach for Learning Reactive Behaviors. In *Real-World Applications of Evolutionary Computing, EvoWorkshops 2000: EvoIASP, EvoSCONDI, EvoTel, EvoSTIM, EvoROB, and EvoFlight*, pages 347--356, London, UK. Springer-Verlag.

[Crain et al., 1999] Crain, T., Bishop, R., Fowler, W., and Rock, K. (1999). Optimal interplanetary trajectory design via hybrid genetic algorithm/recursive quadratic program search. In *Ninth AAS/AIAA Space Flight Mechanics Meeting*, pages 449--466, Breckenridge CO.

[Cramer, 1985] Cramer, N. (1985). A Representation for the Adaptive Generation of Simple Sequential Programs. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 183--187, Hillsdale, NJ, USA. L. Erlbaum Associates Inc.

[Czarn et al., 2004] Czarn, A., MacNish, C., Vijayan, K., Turlach, B., and Gupta, R. (2004). Statistical exploratory analysis of genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 8(4):405--421.

[Davidor, 1990a] Davidor, Y. (1990a). Epistasis Variance: A Viewpoint on GA-Hardness. In [Rawlins, 1991b], pages 23--35.

[Davidor, 1990b] Davidor, Y. (1990b). Epistasis variance: Suitability of a representation to genetic algorithms. *Complex Systems*, 4(4):369--383.

[Davidson and Davidson, 1994] Davidson, A. and Davidson, J. (1994). *Stochastic Limit Theory: An Introduction for Econometricians*. Oxford University Press.

[Davis, 1991] Davis, L. (1991). *Handbook of genetic algorithms*. Arden Shakespeare.

[Dawkins, 1986] Dawkins, R. (1986). The Blind Watchmaker (Harlow).

[Dawkins, 1990] Dawkins, R. (1990). *The Selfish Gene*. Oxford University Press.

[De Bonet et al., 1997] De Bonet, J., Isbell, C., and Viola, P. (1997). MIMIC: Finding optima by estimating probability densities. *Advances in Neural Information Processing Systems*, 9:424--430.

[de Garis, 1991] de Garis, H. (1991). Genetic Programming: Building Artificial Nervous Systems with Genetically Programmed Neural Network Modules. *Neural and Intelligent Systems Integration: Fifth and Sixth Generation Integrated Reasoning Information Systems*, page 207.

[De Jong, 1987] De Jong, K. (1987). On using genetic algorithms to search program spaces. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, pages 210--216, Hillsdale, NJ, USA. L. Erlbaum Associates Inc.

[De Jong, 2006] De Jong, K. (2006). *Evolutionary computation : A Unified Approach*. Cambridge, Mass. : MIT Press.

[De Jong, 1975] De Jong, K. A. (1975). *An analysis of the behavior of a class of genetic adaptive systems.* PhD thesis, Ann Arbor, MI, USA.

[Deb et al., 2002] Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6:182--197.

[Dill et al., 1995] Dill, K., Bromberg, S., Yue, K., Fiebig, K., Yee, D., Thomas, P., and Chan, H. (1995). Principles of Protein Folding - A Perspective from Simple Exact Models. *Protein Science*, 5:561--602.

[Dixon and Szegö, 1978] Dixon, L. and Szegö, G. (1978). *Towards Global Optimisation*, chapter The Global Optimization Problem: An Introduction, pages 1--15. Number 2. North-Holland.

[Doran and Michie, 1966] Doran, J. and Michie, D. (1966). Experiments with the graph traveser program. In *Proceedings of the Royal Society of London (A)*, volume 294, pages 235 -- 259.

[Durillo et al., 2006] Durillo, J., Nebro, A., Luna, F., Dorronsoro, B., and Alba, E. (2006). jMetal: a Java Framework for Developing Multi-Objective Optimization Metaheuristics. Technical Report ITI-2006-10, Departamento de Lenguajes y Ciencias de la Computación, University of Málaga.

[Eiben and Jelasity, 2002] Eiben, A. and Jelasity, M. (2002). A Critical Note on Experimental Research Methodology In EC. In *In Proceedings of the 2002 Congress on Evolutionary Computation (CEC 2002)*, pages 582--587. IEEE Press.

[Eiben et al., 2007] Eiben, A., Michalewicz, Z., Schoenauer, M., and Smith, J. (2007). Parameter control in evolutionary algorithms. *Parameter Setting in Evolutionary Algorithms*, 3(2):19--46.

[Eiben and Schippers, 1998] Eiben, A. E. and Schippers, C. A. (1998). On evolutionary exploration and exploitation. *Fundamenta Informaticae*, 35(1-4):35--50.

[Engelbrecht, 2007] Engelbrecht, A. (2007). *Computational Intelligence: An Introduction*. Wiley Publishing.

[Eshelman and Schaffer, 1992] Eshelman, L. J. and Schaffer, J. D. (1992). Real-Coded Genetic Algorithms and Interval-Schemata. In *Foundations of Genetic Algorithms*, pages 187--202.

[Espejo et al., 2010a] Espejo, P., Ventura, S., and Herrera, F. (2010a). A Survey on the Application of Genetic Programming to Classification. *IEEE Transaction on Systems, Man and Cybernetics - Part C: Applications and Reviews*, 40(2):121.

[Espejo et al., 2010b] Espejo, P. G., Ventura, S., and Herrera, F. (2010b). A survey on the application of genetic programming to classification. *IEEE Transaction on Systems, Man and Cybernetics - Part C: Applications and Reviews*, 40(2):121--144.

[Etxeberria, R. and Larrañaga, P., 1999] Etxeberria, R. and Larrañaga, P. (1999). Global optimization using Bayesian networks. In *Proceedings of the Second Symposium on Artificial Intelligence (CIMAF-99)*, pages 151--173.

[Fidanova et al., 2009] Fidanova, S., Alba, E., and Molina, G. (2009). Memetic Simulated Annealing for the GPS Surveying Problem. In *Numerical Analysis and Its Applications: 4th International Conference, NAA 2008, Lozenetz, Bulgaria, June 16-20, 2008. Revised Selected Papers*, pages 281--288, Berlin, Heidelberg. Springer-Verlag.

[Fisher, 1966] Fisher, S. (1966). *The design of experiments*. Oliver & Boyd.

[Fober et al., 2009] Fober, T., Mernberger, M., Klebe, G., and Hullermeier, E. (2009). Evolutionary construction of multiple graph alignments for the structural analysis of biomolecules. *Bioinformatics*, 25(16):2110.

[Fogel, 1995] Fogel, D. (1995). *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press, Piscataway, NJ, USA.

[Fogel, 2000] Fogel, D. (2000). Applying Fogel and Burgin's ``Competitive goal-seeking through Evolutionary Programming'' to coordination, trust, and bargaining games. In *Proceedings of the 2000 Congress on Evolutionary Computation, 2000*, volume 2, pages 1210--1216.

[Fogel et al., 1990] Fogel, D., Fogel, L., and Atmar, J. (1990). Evolutionary Programming for Training Neural Networks. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, volume 1, pages 601--605.

[Fogel et al., 1991] Fogel, D., Fogel, L., and Atmar, J. (1991). Meta-Evolutionary Programming. In *Proceedings of the Twenty-Fifth Conference on Signals, Systems and Computers, 1991*, volume 1, pages 540--545.

[Fogel et al., 1996] Fogel, D., Ghozeil, A., Inc, N., and La Jolla, C. (1996). Using Fitness Distributions to Design more Efficient Evolutionary Computations. In *Proceedings of IEEE International Conference on Evolutionary Computation, 1996*, pages 11--19.

[Fogel, 1999] Fogel, L. (1999). *Intelligence through simulated evolution: forty years of evolutionary programming*. John Wiley & Sons, Inc., New York, NY, USA.

[Fogel et al., 1966] Fogel, L. J., Owens, A. J., and Walsh, M. J. (1966). *Artificial Intelligence through Simulated Evolution*. John Wiley, New York, USA.

[Fontana et al., 1993] Fontana, W., Stadler, P., Bornberg-Bauer, E., Griesmacher, T., Hofacker, I., Tacker, M., Tarazona, P., Weinberger, E., and Schuster, P. (1993). RNA folding and combinatory landscapes. *Physical Review E*, 47(3):2083--2099.

[Franca et al., 1999] Franca, P., Mendes, A., Mendes, R., Moscato, P., and Unicamp, C. (1999). Memetic Algorithms To Minimize Tardiness On A Single Machine with Sequence-Dependent Setup. In *In Proceedings of the 5th International Conference of the Decision Sciences Institute*, pages 1708--1710.

[Franklin and Bergerman, 2000] Franklin, B. and Bergerman, M. (2000). Cultural algorithms: Concepts and experiments. In *Proceedings of the 2000 Congress on Evolutionary Computation*, pages 1245--1251.

[Freitas, 2001] Freitas, A. (2001). A Survey of Evolutionary Algorithms for Data Mining and Knowledge Discovery. In *Advances in Evolutionary Computation*, pages 819--845. Springer-Verlag.

[Fujiko and Dickinson, 1987] Fujiko, C. and Dickinson, J. (1987). Using the genetic algorithm to generate LISP source code to solve the prisoner's dilemma. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, pages 236--240, Hillsdale, NJ, USA. L. Erlbaum Associates Inc.

[Gallagher and Yuan, 2006] Gallagher, M. and Yuan, B. (2006). A general-purpose tunable landscape generator. *IEEE Transactions on Evolutionary Computation*, 10(5):590--603.

[García et al., 2007] García, S., Molina, D., Lozano, M., and Herrera, F. (2007). Un estudio experimental sobre el uso de test no paramétricos para analizar el comportamiento de los algoritmos evolutivos en problemas de optimización. In *Actas del Quinto Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados. MAEB 2007.*, Puerto de La Cruz, Tenerife.

[García et al., 2009] García, S., Molina, D., Lozano, M., and Herrera, F. (2009). A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 Special Session on Real Parameter Optimization. *Journal of Heuristics*, 15(6):617--644.

[García-Pelayo and Stadler, 1997] García-Pelayo, R. and Stadler, P. F. (1997). Correlation length, isotropy and meta-stable states. In *Proceedings of the 16th annual international conference of the Center for Nonlinear Studies on Landscape paradigms in physics and biology : concepts, structures and dynamics*, pages 240--254, Amsterdam, The Netherlands, The Netherlands. Elsevier Science Publishers B. V.

[Garnier and Kallel, 1999] Garnier, J. and Kallel, L. (1999). How to detect all maxima of a function? In *Proceedings of the Second EVONET Summer School on Theoretical Aspects of Evolutionary Computing*, pages 343--370.

[Garnier and Kallel, 2000] Garnier, J. and Kallel, L. (2000). Statistical Distribution of the Convergence Time of Evolutionary Algorithms for Longpath Problems.

[Garnier and Kallel, 2002] Garnier, J. and Kallel, L. (2002). Efficiency of Local Search with Multiple Local Optima. *SIAM J. Discret. Math.*, 15(1):122--141.

[Gen and Lin, 2008] Gen, M. and Lin, L. (2008). Applications of Evolutionary Technology to Manufacturing and Logistics Systems: State-of-the Art Survey. *IEEJ Transactions on Electronics, Information and Systems*, 128:346--351.

[Glover and Taillard, 1993] Glover, F. and Taillard, E. (1993). A user's guide to tabu search. *Annals of Operations Research*, 41(1):1--28.

[Goldberg, 1989a] Goldberg, D. (1989a). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional.

[Goldberg et al., 1989] Goldberg, D., Korb, B., Deb, K., et al. (1989). Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, 3(5):493--530.

[Goldberg, 1987] Goldberg, D. E. (1987). *Genetic Algorithms and Simulated Annealing*, chapter Simple genetic algorithms and the minimal, deceptive problem, pages 74--88. Morgan Kauffman.

[Goldberg, 1989b] Goldberg, D. E. (1989b). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Goldberg and Deb, 1991] Goldberg, D. E. and Deb, K. (1991). A Comparative Analysis of Selection Schemes Used in Genetic Algorithms. In Rawlins, G. J. E., editor, *Foundations of Genetic Algorithms*, pages 69--93. San Francisco, CA: Morgan Kaufmann.

[Gottlieb et al., 2002] Gottlieb, J., Marchiori, E., and Rossi, C. (2002). Evolutionary algorithms for the satisfiability problem. *Evolutionary Computation*, 10(1):35--50.

[Grefenstette et al., 1990] Grefenstette, J., Ramsey, C., and Schultz, A. (1990). Learning sequential decision rules using simulation models and competition. *Machine Learning*, 5(4):355--381.

[Grefenstette, 1995] Grefenstette, J. J. (1995). Predictive Models Using Fitness Distributions of Genetic Operators. In *Foundations of Genetic Algorithms 3*, pages 139--161. Morgan Kaufmann.

[Hancock, 1994] Hancock, P. J. B. (1994). An Empirical Comparison of Selection Methods in Evolutionary Algorithms. In *Selected Papers from AISB Workshop on Evolutionary Computing*, pages 80--94, London, UK. Springer-Verlag.

[Hansen, 2006] Hansen, N. (2006). The CMA Evolution Strategy: A Comparing Review. *Towards a new Evolutionary Computation*, pages 75--102.

[Hansen, 2007] Hansen, N. (2007). *The CMA Evolution Strategy: A Tutorial*.

[Hansen and Kern, 2004] Hansen, N. and Kern, S. (2004). Evaluating the CMA evolution strategy on multimodal test functions. In *Parallel Problem Solving from Nature-PPSN VIII*, pages 282--291. Springer.

[Hansen et al., 2003] Hansen, N., Müller, S., and Koumoutsakos, P. (2003). Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary Computation*, 11(1):1--18.

[Hansen et al., 2009] Hansen, N., Niederberger, A., Guzzella, L., and Koumoutsakos, P. (2009). A method for handling uncertainty in evolutionary optimization with an application to feedback control of combustion. *IEEE Transactions on Evolutionary Computation*, 13(1):180--197.

[Hansen and Ostermeier, 1996] Hansen, N. and Ostermeier, A. (1996). Adapting Arbitrary Normal Mutation Distributions in Evolution Strategies: the Covariance Matrix Adaptation. In *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*, pages 312--317.

[Hansen and Ostermeier, 2001] Hansen, N. and Ostermeier, A. (2001). Completely Derandomized Self-Adaptation in Evolution Strategies. *Evolutionary Computation*, 9:159--195.

[Harding and Miller, 2006] Harding, S. and Miller, J. (2006). The dead state: A comparison between developmental and direct encodings. In *Proc. GECCO Workshop on Complexity Through Development and Self-Organizing Representations*.

[Harik et al., 1999] Harik, G., Lobo, F., and Goldberg, D. (1999). The compact genetic algorithm. *IEEE Transactions on Evolutionary Computation*, 3(4):287--297.

[Harp et al., 1989] Harp, S., Samad, T., and Guha, A. (1989). Towards the genetic synthesis of neural network. In *Proceedings of the third international conference on Genetic algorithms*, pages 360--369, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Hart et al., 2004] Hart, W., Krasnogor, N., and Smith, J. (2004). Memetic evolutionary algorithms. *Recent advances in memetic algorithms*, pages 3--27.

[Heckendorn and Whitley, 1997] Heckendorn, R. and Whitley, D. (1997). A Walsh analysis of NK-landscapes. In *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 41--48.

[Heckendorn and Whitley, 1999] Heckendorn, R. and Whitley, D. (1999). Predicting epistasis from mathematical models. *Evolutionary Computation*, 7(1):69--101.

[Heckendorn et al., 1997] Heckendorn, R., Whitley, D., and Rana, S. (1997). Nonlinearity, hyperplane ranking and the simple genetic algorithm. *Foundations of Genetic Algorithms*, 4:181--202.

[Herrera et al., 2003] Herrera, F., Lozano, M., and Sánchez, A. (2003). A taxonomy for the crossover operator for real-coded genetic algorithms: An experimental study. *International Journal of Intelligence Systems*, 18(3):309--338.

[Herrera et al., 1998] Herrera, F., Lozano, M., and Verdegay, J. L. (1998). Tackling Real-Coded Genetic Algorithms: Operators and Tools for Behavioural Analysis. *Artificial Intelligence Review*, 12:265--319.

[Hildebrand et al., 1999] Hildebrand, L., Reusch, B., and Fathi, M. (1999). Directed mutation - a new self-adaptation for evolution strategies. In *In Proceedings of the 1999 Congress on Evolutionary Computation CEC99*, pages 1550 -- 1557.

[Holland, 1975] Holland, J. H. (1975). *Adaptation in natural and artificial systems*. MIT Press, Cambridge, MA, USA.

[Holland, 1976] Holland, J. H. (1976). *Progress in theoretical biology*, volume IV, chapter Adaptation, pages 263--293. New York: Academic Press.

[Hooker, 1995] Hooker, J. N. (1995). Testing heuristics: We have it all wrong. *Journal of Heuristics*, 1:33--42.

[Horn and Goldberg, 1994] Horn, J. and Goldberg, D. (1994). *Foundations of Genetic Algorithms*, volume 3, chapter Genetic Algorithm Difficulty and the Modality of Fitness Landscapes, pages 243--269. Morgan Kaufmann.

[Horn et al., 1994] Horn, J., Goldberg, D., and Deb, K. (1994). Long path problems. In *Parallel Problem Solving from Nature: PPSN III*, pages 149--158. Springer.

[Huang et al., 2007] Huang, V., Qin, A., Deb, K., Zitzler, E., Suganthan, P., Liang, J., Preuss, M., and Huband, S. (2007). Problem Definitions for Performance Assessment of Multi-objective Optimization Algorithms, Special Session on Constrained Real-Parameter Optimization. Technical Report, Nanyang Technological University.

[Ichimura and Kuriyama, 1998] Ichimura, T. and Kuriyama, Y. (1998). Learning of neural networks with parallel hybrid GA using a royalroad function. In *Neural Networks Proceedings, 1998. IEEE World Congress on Computational Intelligence. The 1998 IEEE International Joint Conference on*, volume 2.

[Inza and Sierra, 2000] Inza, I. and Sierra, B. (2000). Feature weighting for nearest neighbor by estimation of bayesian networks algorithms. Technical report, University of the Basque Country.

[Jin and Reynolds, 2000] Jin, X. and Reynolds, R. (2000). Mining Knowledge in Large Scale Databases Using Cultural Algorithms with Constraint Handling Mechanisms. In *Proceedings of the Congress on Evolutionary Computation 2000 (CEC'2000)*, volume 2, pages 1498--1506.

[Jones, 1995] Jones, T. (1995). *Evolutionary algorithms, fitness landscapes and search*. PhD thesis, University of New Mexico, Albuquerque, NM.

[Jones and Forrest, 1995] Jones, T. and Forrest, S. (1995). Fitness Distance Correlation as a Measure of Problem Difficulty for Genetic Algorithms. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 184--192. Morgan Kaufmann.

[Joshi and Sanderson, 1997] Joshi, R. and Sanderson, A. C. (1997). Minimal representation multisensor fusion using differential evolution. In *CIRA '97: Proceedings of the 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, page 266, Washington, DC, USA. IEEE Computer Society.

[Juang et al., 2008] Juang, J.-G., Huang, M.-T., and Liu, W.-K. (2008). PID Control Using Presearched Genetic Algorithms for a MIMO System. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 38(5):716--727.

[Kallel et al., 2001] Kallel, L., Naudts, B., and Reeves, C. (2001). *Theoretical aspects of evolutionary computing*, chapter Properties of fitness functions and search landscapes, pages 175 -- 206. Springer-Verlag, London, UK.

[Kallel et al., 1999] Kallel, L., Naudts, B., and Schoenauer, M. (1999). On functions with a given fitness-distance relation. In *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99*, volume 3, pages 1910--1916.

[Kalyanmoy, 1997] Kalyanmoy, D. (1997). Deceptive Landscapes. *Handbook of Evolutionary Computation*, 2:2.7.1--2.7.5.

[Kauffman, 1993] Kauffman, S. A. (1993). *The Origins of Order: Self-Organization and Selection in Evolution*. Oxford University Press.

[Kennedy, 1997] Kennedy, J. (1997). Continuous valued multimodality generator for evolutionary algorithms. [Online]. Available: http://www.cs.uwyo.edu/ wspears/multi.kennedy.html.

[Kennedy and Spears, 1998] Kennedy, J. and Spears, W. M. (1998). Matching Algorithms to Problems: An Experimental Test of the Particle Swarm and Some Genetic Algorithms on the Multimodal Problem Generator. In *In Proceedings of the IEEE Congress on Evolutionary Computation (CEC 1998)*, pages 78--83.

[Kollman et al., 1997] Kollman, K., Miller, J. H., and Page, S. E. (1997). Political Institutions and Sorting in a Tiebout Model. *American Economic Review*, 87(5):977--92.

[Koza, 1990] Koza, J. (1990). Genetic programming: a paradigm for genetically breeding populations of computer programs to solve problems. Technical report, Stanford University, Stanford, CA, USA.

[Koza, 1992a] Koza, J. (1992a). Genetic evolution and co-evolution of game strategies. In *International Conference on Game Theory and Its Applications*.

[Koza, 1992b] Koza, J. (1992b). *Genetic Programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems)*. The MIT Press.

[Koza, 1994] Koza, J. (1994). *Genetic programming II : automatic discovery of reusable programs*. Complex adaptive systems. MIT Press.

[Koza et al., 1999] Koza, J., H.B., F., Andre, D., and Keane, M. (1999). *Genetic Programming III: Darwinian Invention and Problem Solving*. Morgan Kaufmann.

[Koza et al., 2003] Koza, J., Keane, M., Streeter, M., Mydlowec, W., Yu, J., and Lanza, G. (2003). *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers.

[Krakhofer and Stadler, 1996] Krakhofer, B. and Stadler, P. F. (1996). Local Minima in the Graph Bipartitioning Problem. Working Papers 96-02-005, Santa Fe Institute.

[Kurup et al., 2003] Kurup, D., Himdi, M., and Rydberg, A. (2003). Synthesis of uniform amplitude unequally spaced antenna arrays using the differential evolution algorithm. *IEEE Transactions on Antennas and Propagation*, 51(9):2210--2217.

[Kyprianou et al., 2001] Kyprianou, A., Worden, K., and Panet, M. (2001). Identification of Hysteretic Systems Using the Differential Evolution Algorithm. *Journal of Sound and Vibration*, 248(2):289 -- 314.

[Larrañaga et al., 2003] Larrañaga, P., Lozano, J., and Mühlenbein, H. (2003). Algoritmos de Estimación de Distribuciones en Problemas de Optimización Combinatoria. *Inteligencia Artificial, Revista Iberoamericana de IA*, 7(19):149--168.

[Li et al., 2008] Li, C., Yang, S., Nguyen, T., Yao, X., Jin, Y., Beyer, H.-G., and Suganthan, P. (2008). Benchmark Generator for CEC 2009 Competition on Dynamic Optimization. Technical Report, University of Leicester, University of Birmingham, Nanyang Technological University.

[Li and Yong, 2009] Li, J. and Yong, J. (2009). Estimation of Distribution Algorithms for Job Schedule Problem. In *ICIC '09: Proceedings of the 2009 Second International Conference on Information and Computing Science*, pages 7--10, Washington, DC, USA. IEEE Computer Society.

[Li et al., 2007] Li, M., Goldberg, D., Sastry, K., and Yu, T. (2007). Real-coded ECGA for solving decomposable real-valued optimization problems. In *IEEE Congress on Evolutionary Computation, 2007. CEC 2007*, pages 2194--2201.

[Li et al., 2003] Li, Y., Rao, L., He, R., Xu, G., Wu, Q., Ge, M., and Yan, W. (2003). Image reconstruction of EIT using differential evolution algorithm. In *Proceedings of the 25th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, 2003*, volume 2.

[Liang et al., 2006] Liang, J. J., Runarsson, T. P., Mezura-Montes, E., Clerc, M., Suganthan, P. N., Coello, C. A. C., and Deb, K. (2006). Problem Definitions and Evaluation Criteria for the CEC 2006 Special Session on Constrained Real-Parameter Optimization. Technical report, Nanyang Technological University.

[Locatelli, 2003] Locatelli, M. (2003). A note on the Griewank test function. *Journal of Global Optimization*, 25(2):169--174.

[Lopez Cruz et al., 2003a] Lopez Cruz, I., Van Willigenburg, L., and Van Straten, G. (2003a). Efficient differential evolution algorithms for multimodal optimal control problems. *Applied Soft Computing*, 3(2):97--122.

[Lopez Cruz et al., 2003b] Lopez Cruz, I., Van Willigenburg, L., and Van Straten, G. (2003b). Optimal control of nitrate in lettuce by a hybrid approach: differential evolution and adjustable control weight gradient algorithms. *Computers and Electronics in Agriculture*, 40(1-3):179--197.

[Magoulas et al., 2001] Magoulas, G., Plagianakos, V., and Vrahatis, M. (2001). Hybrid methods using evolutionary algorithms for on-line training. In *INNS--IEEE International Joint Conference on Neural Networks (IJCNN), July 14--19, Washington, DC, USA*, volume 3, pages 2218--2223.

[Magoulas et al., 2004] Magoulas, G., Plagianakos, V., and Vrahatis, M. (2004). Neural network-based colonoscopic diagnosis using on-line learning and differential evolution. *Applied Soft Computing*, 4(4):369--379.

[Mallipeddi and Suganthan, 2010] Mallipeddi, R. and Suganthan, P. (2010). Problem Definitions and Evaluation Criteria for the CEC 2010 Competition on Constrained Real-Parameter Optimization. Technical Report, Nanyang Technological University, Singapore.

[Manderick et al., 1991] Manderick, B., de Weger, M. K., and Spiessens, P. (1991). The Genetic Algorithm and the Structure of the Fitness Landscape. In Belew, R. K. and Booker, L. B., editors, *ICGA*, pages 143--150. Morgan Kaufmann.

[Marín, 2005] Marín, J. (2005). *Algoritmos Macroevolutivos: desde la simulación hasta el modelo estocástico*. PhD thesis, Universitat Politecnica de Catalunya.

[Marin and Sole, 1998] Marin, J. and Sole, R. V. (1998). Macroevolutionary Algorithms: A New Optimization Method on Fitness Landscapes. Working papers, Santa Fe Institute.

[Merz and Freisleben, 2000] Merz, P. and Freisleben, B. (2000). Fitness Landscapes and Memetic Algorithms and Greedy Operators for Graph Bi-Partitioning. *Evolutionary Computation*, 8(1):61 -- 91.

[Mezard et al., 1987] Mezard, M., Parisi, G., and Virasoro, M. (1987). *Spin Glass Theory and Beyond*, volume 9. World Scientific Publishing Company.

[Mezura-Montes, 2009] Mezura-Montes, E. (2009). *Constraint-Handling in Evolutionary Optimization*. Springer Publishing Company, Incorporated.

[Mezura-Montes et al., 2006] Mezura-Montes, E., Velázquez-Reyes, J., Coello, C., et al. (2006). Modified differential evolution for constrained optimization. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pages 25--32. IEEE.

[Michalewicz, 1994] Michalewicz, Z. (1994). *Genetic Algorithms Plus Data Structures Equals Evolution Programs*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[Michalewicz, 1997] Michalewicz, Z. (1997). *Evolutionary Algorithms in Engineering Applications*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[Michalewicz et al., 2000] Michalewicz, Z., Deb, K., Schmidt, M., and Stidsen, T. (2000). Testcase Generator for Nonlinear Continuous Parameter Optimization Techniques. *IEEE Transactions on Evolutionary Computation*, 4:197--215.

[Moreno-Pérez et al., 2007] Moreno-Pérez, J., Campos-Rodríguez, C., and Laguna, M. (2007). Sobre la Comparación de Metaheurísticas mediante Técnicas Estadísticas no Paramétricas. In *Actas del Quinto Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados. MAEB 2007.*, Puerto de La Cruz, Tenerife.

[Morrison, 2004] Morrison, R. (2004). *Designing evolutionary algorithms for dynamic environments*. Natural Computer Series. Springer-Verlag New York Inc.

[Morrison and De Jong, 1999] Morrison, R. and De Jong, K. (1999). A test problem generator for non-stationary environments. In *Congress on Evolutionary Computation*, volume 3, pages 2047--2053.

[Moscato, 1989] Moscato, P. (1989). On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *Caltech Concurrent Computation Program, C3P Report*, 826:1989.

[Moscato and Cotta, 2003] Moscato, P. and Cotta, C. (2003). An introduction to memetic algorithms. *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial*, 19:131--148.

[Moscato and Norman, 1992] Moscato, P. and Norman, M. (1992). A memetic approach for the traveling salesman problem implementation of a computational ecology for combinatorial optimization on message-passing systems. *Parallel Computing and Transputer Applications*, -:177--186.

[Mosk-Aoyama and Shah, 2006] Mosk-Aoyama, D. and Shah, D. (2006). Computing separable functions via gossip. In *PODC '06: Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, pages 113--122, New York, NY, USA. ACM.

[Mühlenbein et al., 1999] Mühlenbein, H., Mahnig, T., and Ochoa, A. (1999). Schemata, distributions and graphical models in evolutionary optimization. *Journal of Heuristics*, 5(2):213--247.

[Mühlenbein and Paasz, 1996] Mühlenbein, H. and Paasz, G. (1996). From recombination of genes to the estimation of distributions. Binary parameters. *Lecture Notes in Computer Science: Parallel Problem Solving from Nature, PPSN IV*, 1411:178--187.

[Mühlenbein and Schlierkamp-Voosen, 1993] Mühlenbein, H. and Schlierkamp-Voosen, D. (1993). Predictive models for the breeder genetic algorithm in continuous parameter optimization. *Evolutionary Computation*, 1(1):25--49.

[Nannen and Eiben, 2007] Nannen, V. and Eiben, A. (2007). Relevance estimation and value calibration of evolutionary algorithm parameters. In *Proc. IJCAI*, volume 7, pages 975--980.

[Naudts, 1998] Naudts, B. (1998). *Measuring GA-hardness*. Phd Thesis, Universitaire Instelling Antwerpen and Departement Wiskunde-informatica .

[Naudts and Verschoren, 1999] Naudts, B. and Verschoren, A. (1999). Epistasis and Deceptivity. In *Simon Stevin - Bulletin of the Belgian Mathematical Society , 6 , 147-154. Novkovic, S.*

[Naujoks et al., 2006] Naujoks, B., Quagliarella, D., and Bartz-Beielstein, T. (2006). Sequential parameter optimisation of evolutionary algorithms for airfoil design. In Winter, G., editor, *Proc. Design and Optimization: Methods and Applications, (ERCOFTAC'06)*, pages 231--235. University of Las Palmas de Gran Canaria.

[Ocenasek and Schwarz, 2002] Ocenasek, J. and Schwarz, J. (2002). Estimation Distribution Algorithm for Mixed Continuous - discrete Optimization Problems. *Intelligent technologies: theory and applications: new trends in intelligent technologies*, 1:227.

[Ochoa et al., 2008] Ochoa, G., Tomassini, M., Verel, S., and Darabos, C. (2008). A study of NK landscapes' basins and local optima networks. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 555--562. ACM.

[Omran et al., 2005] Omran, M., Engelbrecht, A., and Salman, A. (2005). Differential evolution methods for unsupervised image classification. In *In Proceedings of The 2005 Congress of Evolutionary Computation (CEC 2005)*, pages 966--973. IEEE Press.

[Östermark, 1999] Östermark, R. (1999). A Neuro-Genetic Algorithm for Heteroskedastic Time-Series Processes Empirical Tests on Global Asset Returns. *Soft Computing-A Fusion of Foundations, Methodologies and Applications*, 3(4):206--220.

[Ostrowski and Reynolds, 1999] Ostrowski, D. and Reynolds, R. (1999). Knowledge-Based Software Testing Agent using Evolutionaly Learning with Cultural Algorithms. In *Proceedings of the IEEE Congress on Evolutionary Computation*, volume 3.

[Ozcan and Mohan, 1998] Ozcan, E. and Mohan, C. (1998). Steady state memetic algorithm for partial shape matching. In *Evolutionary Programming VII*, pages 527--536. Springer.

[Ozcelik and Erzurumlu, 2006] Ozcelik, B. and Erzurumlu, T. (2006). Comparison of the warpage optimization in the plastic injection molding using ANOVA, neural network model and genetic algorithm. *Journal of Materials Processing Technology*, 171(3):437--445.

[Pahner and Hameyer, 2002] Pahner, U. and Hameyer, K. (2002). Adaptive coupling of differential evolution and multiquadrics approximation for the tuning of the optimization process. *IEEE Transactions on Magnetics*, 36(4):1047--1051.

[Paterlini and Krink, 2004] Paterlini, S. and Krink, T. (2004). High performance clustering with differential evolution. In *In Proceedings of The 2004 Congress on Evolutionary Computation (CEC 2004)*, volume 2.

[Paulinas and Usinskas, 2007] Paulinas, M. and Usinskas, A. (2007). A survey of genetic algorithms applications for image enchancement and segmentation. *Information Technology and Control*, 36(3):278--284.

[Pelikan and Goldberg, 2006] Pelikan, M. and Goldberg, D. (2006). Hierarchical Bayesian optimization algorithm. *Scalable Optimization via Probabilistic Modeling*, 1:63--90.

[Pelikan et al., 2000] Pelikan, M., Goldberg, D., and Cantú-Paz, E. (2000). BOA: The Bayesian optimization algorithm. *Evolutionary Computation*, 8(3):311--340.

[Pelikan and Muhlenbein, 1999] Pelikan, M. and Muhlenbein, H. (1999). The bivariate marginal distribution algorithm. *Advances in Soft Computing - Engineering Design and Manufacturing*, -:521--535.

[Preuss et al., 2007] Preuss, M., Rudolph, G., and Tumakaka, F. (2007). Solving multimodal problems via multiobjective techniques with Application to phase equilibrium detection. In *Proceedings of the International Congress on Evolutionary Computation (CEC2007). Piscataway (NJ): IEEE Press. Im Druck.*

[Price et al., 2005] Price, K., Storn, R., and Lampinen, J. (2005). *Differential Evolution: A Practical Approach to Global Optimization*. Natural Computing Series. Springer-Verlag New York, Inc. Secaucus, NJ, USA.

[Price, 1997] Price, K. V. (1997). Differential Evolution vs. The Functions of the 2nd ICEO. In *In Proceedings of the IEEE International Conference of Evolutionary Computation*, pages 153--157.

[Qian et al., 2008] Qian, B., Wang, L., Hu, R., Wang, W., Huang, D., and Wang, X. (2008). A hybrid differential evolution method for permutation flow-shop scheduling. *The International Journal of Advanced Manufacturing Technology*, 38(7):757--777.

[Quick et al., 1998] Quick, R., Rayward-Smith, V., and Smith, G. (1998). Fitness distance correlation and ridge functions. In *Parallel Problem Solving from Nature - PPSN V*, page 77. Springer.

[Radcliffe and Surry, 1995] Radcliffe, N. J. and Surry, P. D. (1995). Fundamental Limitations on Search Algorithms: Evolutionary Computing in Perspective. In *Lecture Notes in Computer Science 1000*, pages 275--291. Springer-Verlag.

[Rae and Parameswaran, 1998] Rae, A. and Parameswaran, S. (1998). Application-specific heterogeneous multiprocessor synthesis using differential-evolution. In *In Proceedings. 11th International Symposium on System Synthesis (Cat. No.98EX210). IEEE Comput. Soc, Los Alamitos*, pages 83--88.

[Rawlins, 1991a] Rawlins, G. (1991a). *Foundations of Genetic Algorithms*. Morgan Kaufmann.

[Rawlins, 1991b] Rawlins, G. J. E., editor (1991b). *Proceedings of the First Workshop on Foundations of Genetic Algorithms. Bloomington Campus, Indiana, USA, July 15-18 1990*. Morgan Kaufmann.

[Ray, 1994] Ray, T. (1994). An evolutionary approach to synthetic biology: Zen and the art of creating life. *Artificial Life*, 1(1-2):179--209.

[Rechenberg, 1973] Rechenberg, I. (1973). *Evolutionsstrategie : Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Problemata. Frommann-Holzboog, Stuttgart-Bad Cannstatt.

[Reeves, 1993] Reeves, C. (1993). Hybrid Genetic Algorithms for Bin-packing and Related Problems. *Annals of Operations Research*, 63:371--396.

[Reeves and Wright, 1995a] Reeves, C. and Wright, C. (1995a). An experimental design perspective on genetic algorithms. *Foundations of Genetic Algorithms*, 3:7--22.

[Reeves and Wright, 1995b] Reeves, C. and Wright, C. (1995b). Epistasis in genetic algorithms: An experimental design perspective. In *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 217--224.

[Reidys and Stadler, 2001] Reidys, C. M. and Stadler, P. F. (2001). Combinatorial Landscapes. Technical Report 01-03-014, Santa Fe Institute.

[Reynolds, 1999] Reynolds, R. (1999). *New ideas in optimization*, chapter Cultural algorithms: theory and applications, pages 367--378. McGraw-Hill Ltd., UK, Maidenhead, UK, England.

[Reynolds and Peng, 2004] Reynolds, R. and Peng, B. (2004). Cultural Algorithms: Modeling of How Cultures Learn to Solve Problems. In *ICTAI '04: Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence*, pages 166--172, Washington, DC, USA. IEEE Computer Society.

[Reynolds and Rolnick, 1995] Reynolds, R. and Rolnick, S. (1995). Learning the parameters for a gradient-based approach to image segmentation using cultural algorithms. *Symposium on Intelligence in Neural and Biological Systems*, -:240.

[Reynolds, 1979] Reynolds, R. G. (1979). *An adaptive computer model of the evolution of agriculture for hunter-gatherers in the valley of Oaxaca, Mexico*. PhD thesis, University of Michigan, Ann Arbor.

[Reynolds and Al-Shehri, 1998] Reynolds, R. G. and Al-Shehri, H. (1998). The Use of Cultural Algorithms with Evolutionary Programming to Guide Decision Tree Induction in Large Databases. In *IEEE World Congress on Computational Intelligence, Proceedings of the International Conference of Evolutionary Computation*, pages 541 -- 546.

[Reynolds and Chung, 1997] Reynolds, R. G. and Chung, C. (1997). Knowledge-based Self-Adaptation in Evolutionary Programming using Cultural Algorithms. In *Proceedings of the IEEE Congress of Evolutionary Computation*.

[Rochet et al., 1996] Rochet, S., Slimane, M., and Venturini, G. (1996). Epistasis for real encoding in genetic algorithms. In *Australian and New Zealand Conference on Intelligent Information Systems, 1996.*, pages 268--271.

[Rochet et al., 1998] Rochet, S., Venturini, G., Slimane, M., and El Kharoubi, E. (1998). A critical and empirical study of epistasis measures for predicting GA performances: a summary. In *Artificial Evolution*, page 275. Springer.

[Rojas et al., 2002] Rojas, I., González, J., Pomares, H., Merelo, J., Castillo, P., and Romero, G. (2002). Statistical analysis of the main parameters involved in the design of a genetic algorithm. *IEEE Transaction on Systems, Man and Cybernetics - Part C: Applications and Reviews*, 32(1):31.

[Romero and Machado, 2007] Romero, J. and Machado, P., editors (2007). *The Art of Artificial Evolution: A Handbook on Evolutionary Art and Music*. Natural Computing Series. Springer.

[Rönkkönen et al., 2005] Rönkkönen, J., Kukkonen, S., and Price, K. V. (2005). Real-Parameter Optimization with Differential Evolution. In *The 2005 IEEE International Congress on Evolutionary Computation (CEC'05)*, page 12.

[Ros and Hansen, 2008] Ros, R. and Hansen, N. (2008). A Simple Modification in CMA-ES Achieving Linear Time and Space Complexity. In Heidelberg, S. B. ., editor, *Parallel Problem Solving from Nature -- PPSN X*, volume 5199/2008 of *Lecture Notes in Computer Science*, pages 296--305.

[Roure et al., 2001] Roure, J., Larrañaga, P., and Sangüesa, R. (2001). *Estimation of Distribution Algorithms: A new tool for Evolutionary Computation*, chapter Comparing K-Means, GAs and EDAs in partitional clustering, pages 343 -- 355. Genetic Algorithms and Evolutionary Computation. Kluwer Academic Publishers.

[Rychtychyj and Reynolds, 1999] Rychtychyj, N. and Reynolds, R. G. (1999). Using Cultural Algorithms to Improve Performance in Semantic Networks. In *Proceedings of the IEEE International Congress on Evolutionary Computation*, volume 3, pages 1651 -- 1656.

[Rzadca and Franciszek, 2005] Rzadca, K. and Franciszek, S. (2005). Heterogeneous Multiprocessor Scheduling with Differential Evolution. In *In Proceedings of the 2005 Congress on Evolutionary Computation*, page to appear. IEEE Press.

[Salcedo-Sanz et al., 2010] Salcedo-Sanz, S., Naldi, M., Perez-Bellido, A., Portilla-Figueras, J., and Ortíz-García, E. (2010). Evolutionary Optimization of Service Times in Interactive Voice Response Systems. *IEEE Transactions on Evolutionary Computation*, 14(4):602--617.

[Salomon, 1997] Salomon, R. (1997). Raising Theoretical Questions About the Utility of Genetic Algorithms. In *EP '97: Proceedings of the 6th International Conference on Evolutionary Programming VI*, pages 275--284, London, UK. Springer-Verlag.

[Santana et al., 1999] Santana, R., Ponce de Leon, E., and Ochoa, A. (1999). The edge Incident Model. In *Proceedings of the Second Symposium on Artificial Intelligence (CIMAF-99)*, pages 352--359.

[Sarma and De Jong, 1997] Sarma, J. and De Jong, K. (1997). Generation gap methods. *Handbook of Evolutionary Computation*, 2(2):2.7:1 -- 2.7:5.

[Schaffer et al., 1989] Schaffer, J. D., Caruana, R. A., Eshelman, L. J., and Das, R. (1989). A study of control parameters affecting online performance of genetic algorithms for function optimization. In *Proceedings of the third international conference on Genetic algorithms*, pages 51--60, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Schaffer et al., 1990] Schaffer, J. D., Eshelman, L. J., and Offutt, D. (1990). Spurious Correlations and Premature Convergence in Genetic Algorithms. In [Rawlins, 1991b], pages 102--112.

[Schneider et al., 2005] Schneider, G., Wersing, H., Sendhoff, B., and Korner, E. (2005). Evolutionary optimization of a hierarchical object recognition model. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 35(3):426--437.

[Schuster and Stadler, 1993] Schuster, P. and Stadler, P. F. (1993). Landscapes - Complex Optimization Problems and Biopolymer Structures. *Computers & Chemistry*, 18:18--295.

[Schwefel, 1981] Schwefel, H.-P. (1981). *Numerical Optimization of Computer Models*. John Wiley & Sons, Inc., New York, NY, USA.

[Sierra et al., 2001] Sierra, B., Jiménez, E., Inza, I., Larrañaga, P., and Muruzábal, J. (2001). *Estimation of Distribution Algorithms: A new tool for Evolutionary Computation*, chapter Rule Induction by Estimation of Distribution Algorithms, pages 313--322. Genetic Algorithms and Evolutionary Computation. Kluwer Academic Publishers.

[Sims, 1994a] Sims, K. (1994a). Evolving 3D morphology and behavior by competition. *Artificial Life*, 1(4):353--372.

[Sims, 1994b] Sims, K. (1994b). Evolving Virtual Creatures. *Computer Graphics (SIGGRAPH Proceedings)*, -:15--22.

[Smit and Eiben, 2010] Smit, S. K. and Eiben, A. E. (2010). Beating the 'world champion' Evolutionary Algorithm via REVAC Tuning. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2010, Barcelona, Spain, 18-23 July 2010*.

[Smith and Smith, 1995] Smith, R. E. and Smith, J. E. (1995). *Foundations of Genetic Algorithms*, volume 5, chapter An examination of tunable, random search landscapes, pages 165--181. Morgan Kaufmann Publishers Inc.

[Smith, 1983] Smith, S. (1983). Flexible learning of problem solving heuristics through adaptive search. In *Proceedings of the Eighth international joint conference on Artificial intelligence*, volume 1, pages 422--425. Morgan Kaufmann Publishers Inc.

[Stadler, 1998] Stadler, B. (1998). Adaptive Platform Dynamics in Multi-Party Spatial Voting. Working papers, Santa Fe Institute.

[Stadler, 2002] Stadler, P. (2002). Fitness Landscapes. *Biological Evolution and Statistical Physics*, 585:183--204.

[Stadler and Happel, 1999] Stadler, P. and Happel, R. (1999). Random field models for fitness landscapes. *Journal of Mathematical Biology*, 38(5):435--478.

[Stadler and Schnabl, 1992] Stadler, P. and Schnabl, W. (1992). The landscape of the traveling salesman problem. *Physics Letters A*, 161:337--344.

[Stadler, 1995] Stadler, P. F. (1995). Towards a Theory of Landscapes.

[Stickberger, 1968] Stickberger, M. M. (1968). *Genetics*. Collier-McMillan Ltd.

[Storn, 1996a] Storn, R. (1996a). Differential Evolution Design of an IIR-Filter. In *In Proceedings of the IEEE International Conference on Evolutionay Computation*, pages 268--273.

[Storn, 1996b] Storn, R. (1996b). On the Usage of Differential Evolution for Function Optimization. In *In 1996 Biennial Conference of the North American Fuzzy Information Processing Society (NAFIPS 1996)*, pages 519--523, Berkeley.

[Storn and Price, 1995] Storn, R. and Price, K. (1995). Differential Evolution - A simple and efficient adaptive scheme for global optimization over continuous spaces. Technical Report TR-95-012, ICSI - International Computer Science Institute, University of California, Berkeley.

[Storn and Price, 1997] Storn, R. and Price, K. (1997). Differential evolution--a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341--359.

[Stuckman, 1988] Stuckman, B. (1988). A global search method for optimizing nonlinear systems. *IEEE Transactions on Systems, Man and Cybernetics*, 18(6):965--977.

[Suganthan et al., 2005] Suganthan, P. N., Hansen, N., Liang, J. J., Deb, K., Chen, Y. P., Auger, A., and Tiwari, S. (2005). Problem definitions and evaluation criteria for the CEC 2005 Special Session on Real Parameter Optimization. Technical report, Nanyang Technological University.

[Sun et al., 2005] Sun, J., Zhang, Q., and Tsang, E. P. K. (2005). DE/EDA: A new evolutionary algorithm for global optimization. *Information Sciences*, 169(3-4):249 -- 262.

[Sutton et al., 2007] Sutton, A. M., Lunacek, M., and Whitley, L. D. (2007). Differential evolution and non-separability: using selective pressure to focus search. In *GECCO '07: Proceedings of the 9th annual Conference on Genetic and Evolutionary Computation*, pages 1428--1435, New York, NY, USA. ACM.

[Sverdlik et al., 1992] Sverdlik, W., Reynolds, R. G., and Zannoni, E. (1992). HYBAL: A Self-Tuning Algorithm for Concept Learning in Highly Autonomous Systems. In *Proceedings if the Third Annual Conference on Artificial Intelligence, Simulation and Planning in High Autonomy Systems*, pages 15 -- 22.

[Tang et al., 2009] Tang, K., Li, X., Suganthan, P., Yang, Z., and Weise, T. (2009). Benchmark Functions for the CEC'2010 Special Session and Competition on Large Scale Global Optimization. Technical Report, Nature Inspired Computation, Applications Laboratory, USTC, China & Nanyang Technological University.

[Tang et al., 2007] Tang, K., Yao, X., Suganthan, P., Macnish, C., Chen, Y., Chen, C., and Yang, Z. (2007). Benchmark Functions for the CEC 2008 Special Session and Competition on Large Scale Global Optimization. Technical Report, Nature Inspired Computation and Applications Laboratory, USTC, China.

[Tarzjan and Moghaddam, 2007] Tarzjan, M. S. and Moghaddam, H. A. (2007). A Novel Evolutionary Approach for Optimizing Content-Based Image Indexing Algorithms. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 37(1):139--153.

[Thompson et al., 2007] Thompson, E., Paulden, T., and Smith, D. (2007). The Dandelion Code: A new coding of spanning trees for genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 11(1):91--100.

[Trautmann and Mehnen, 2009] Trautmann, H. and Mehnen, J. (2009). Statistical methods for improving multi-objective evolutionary optimization. *International Journal of Computation Intelligence Research (IJCIR)*, 5(2):72--78.

[Tsujimura et al., 2008] Tsujimura, Y., Lin, L., and Gen, M. (2008). Applications of Evolutionary Technology to Information and Communication Systems: State-of-the Art Survey. *IEEJ Transactions on Electronics, Information and Systems*, 128:340--345.

[Tsutsui, 2002] Tsutsui, S. (2002). Probabilistic model-building genetic algorithms in permutation representation domain using edge histogram. In *Parallel Problem Solving from Nature - PPSN VII*, pages 224--233. Springer.

[Turing, 1950] Turing, A. M. (1950). Computing Machinery and Intelligence. *Mind*, LIX:433--460.

[Walter, 1953] Walter, W. (1953). *The living brain*. WW Norton New York.

[Wang and Li, 2008] Wang, Y. and Li, B. (2008). Understand behavior and performance of Real Coded Optimization Algorithms via NK-linkage model. In *IEEE Congress on Evolutionary Computation*, pages 801--808. IEEE.

[Weinberger, 1990] Weinberger, E. (1990). Correlated and uncorrelated fitness landscapes and how to tell the difference. *Biological Cybernetics*, 63(5):325--336.

[Weinberger, 1991] Weinberger, E. D. (1991). Local Properties of Kauffman's NK Model: A Tunably Rugged Energy Landscape. *Physical Review A*, 44(10):6399–6413.

[Weise et al., 2008] Weise, T., Niemczyk, S., Skubch, H., Reichle, R., and Geihs, K. (2008). A tunable model for multi-objective, epistatic, rugged, and neutral fitness landscapes. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 795--802. ACM.

[Whitley et al., 1995] Whitley, L. D., Mathias, K. E., Rana, S. B., and Dzubera, J. (1995). Building Better Test Functions. In *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 239--247, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Whitley et al., 1996] Whitley, L. D., Rana, S. B., Dzubera, J., and Mathias, K. (1996). Evaluating evolutionary algorithms. *Artificial Intelligence*, 85(1-2):245--276.

[Wolpert and Macready, 1997] Wolpert, D. H. and Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67--82.

[Wolpert et al., 1995] Wolpert, D. H., Macready, W. G., H, D., and G, W. (1995). No Free Lunch Theorems for Search.

[Wright, 1932] Wright, S. (1932). The roles of mutation, inbreeding, crossbreeding, and selection in evolution. *Proceedings of the Sixth International Congress on Genetics*, 1:356--366.

[Wu and Handroos, 2000] Wu, H. and Handroos, H. (2000). Utilization of differential evolution in inverse kinematics solution of a parallel redundant manipulator. In Howlett, R. J. and Jain, L. C., editors, *Knowledge-Based Intelligent Engineering Systems and Allied Technologies, 2000. Proceedings. Fourth International Conference on* , pages 812--815. IEEE.

[Wu et al., 2010] Wu, Y., Wang, Y., and Liu, X. (2010). Multi-population Based Univariate Marginal Distribution Algorithm for Dynamic Optimization Problems. *Journal of Intelligent and Robotic Systems*, 59(2):1--18.

[Yang et al., 2011] Yang, L., Gao, Z., and Li, K. (2011). Railway freight transportation planning with mixed uncertainty of randomness and fuzziness. *Applied Soft Computing*, 11(1):778--792.

[Yao et al., 2003] Yao, X., Liu, Y., and Liang, K.-H. (2003). Fast Evolutionary Algorithms. In *Proc. Advances Evol. Computing: Theory Applicat.*, page 45 94, New York.

[Yao et al., 1999] Yao, X., Liu, Y., and Lin, G. (1999). Evolutionary Programming Made Faster. *IEEE Transactions on Evolutionary Computation*, 3:21.

[Yeh, 2000] Yeh, W.-C. (2000). A Memetic Algorithm for the min *k*-cut problem. *Computer Aided Civil and Infrastructure Engineering*, 28(2):47 -- 55.

[Ying-ping Chen, 2010] Ying-ping Chen, editor (2010). *Exploitation of Linkage Learning in Evolutionary Algorithms*, volume 3 of *Adaptation Learning and Optimization*. Springer Berlin Heidelberg.

[Ying-ping Chen and Meng-Hiot Lim, 2008] Ying-ping Chen and Meng-Hiot Lim, editor (2008). *Linkage in Evolutionary Computation*, volume 157 of *Studies in Computational Intelligence*. Springer Berlin / Heidelberg.

[Zaharie, 2009] Zaharie, D. (2009). Influence of crossover on the behavior of Differential Evolution Algorithms. *Applied Soft Computing*, 9:1126--1138.

[Zhang and Sanderson, 2009] Zhang, J. and Sanderson, A. (2009). JADE: Adaptive Differential Evolution With Optional External Archive. *IEEE Transactions on Evolutionary Computation*, 13(5):945 -- 958.

[Zhang et al., 2008] Zhang, Q., Zhou, A., Zhao, S., Suganthan, P., Liu, W., and Tiwari, S. (2008). Multiobjective Optimization Test Instances for the CEC 2009 Special Session and Competition. Technical Report CES-887, University of Essex and Nanyang Technological University.

Índice de figuras

Índice de tablas

Glosario

G_{NE} Error genotípico. 93, 103, 105, 109, 111, 112, 119, 125, 137, 144, 145, 148, 153, 166

AE Algoritmo evolutivo, en inglés *Evolutionary Algorithms*. 9, 10, 16, 19, 21--34, 36--43, 46--48, 51, 53--56, 58, 59, 61, 69, 80, 86, 90, 94--97, 99--101, 104, 105, 113, 114, 117, 123, 155, 159, 163--167

AES Número medio de evaluaciones, del inglés *Average number of Evaluations to a Solution*. 39

AG Algoritmo genético, en inglés *Genetic Algorithm*. 16, 17, 20, 26, 32, 33, 40, 54, 56--58, 61, 98, 106, 107, 131, 164

ANOVA Análisis de varianza. 40

BMDA *Bivariate Marginal Distribution Algorithm*. 27

BOA *Bayesian Optimization Algorithm*. 27

CE Computación Evolutiva. 13--15, 19, 20, 22, 28, 30, 33, 37, 41, 47, 53, 61, 124, 161, 163, 167

CEC Congreso de computación evolutiva, del ingés *Congress of Evolutionary Computation*. 2, 34

cGA *Compact Genetic Algorithm*. 27

CMA-ES Covariance Matrix Adaptation - Evolutionary Strategy. 105, 122--125, 127--134, 138, 145, 159--161, 167

CPEM Medida combinada de error y rendimiento, del inglés, Combined Performance and Error Measure. 91--95, 103--105, 109--112, 119, 125, 126, 133, 137, 143--145, 147--150, 153, 159, 160, 166

DE Evolución Diferencial, del inglés *Differential Evolution*. 24, 25, 32, 39, 105, 136, 137, 139--146, 159--161

DoE Diseño de experimentos, del inglés *Design of Experiments*. 41

EBNA *Estimation of Bayesian Network Algorithm*. 27

EcGA *Extended Compact Genetic Algorithm*. 27

EDA Estimation of Distribution Algorithms, Algoritmos de Estimación de Distribuciones. 25--28, 42, 124

EE Estrategias evolutivas, en inglés *Evolutionary Strategies*. 17--20, 24, 26, 33, 58, 124

FDA *Factorized Distribution Algorithm*. 27

FDC Fitness Distance Correlation. 56, 57

FEs Número de llamadas a la función de calidad. 92, 117, 119, 155, 157, 158, 160

FSM Máquinas de estados finitos, en inglés *Finite State Machine*. 18

GECCO *The Genetic and Evolutionary Computation Conference*. 2

GII Grupo Integrado de Ingeniería. 33

hBOA *Hierarchical Bayesian Optimization Algorithm*. 27

JEAF Java Evolutionary Algorithm Framework. 103, 107, 125, 137, 148, 170, 171, 188

MA Algoritmos Macroevolutivos, del inglés *Macroevolutionary Algorithms*. 105, 148, 151--155, 157, 159--161

MBF Mejor calidad media, del inglés *Mean Best Fitness*. 39

MIMIC *Mutual-Information-Maximizing Input Clustering*. 27

NFL Teorema *No-Free-Lunch*. 30, 34

NIaH Problemas tipo aguja en un pajar, del inglés *needle-in-a-haystack*. 51, 58

PBIL *Populaton-based incremental learning*. 27

PE Programación Evolutiva, en inglés *Evolutionary Programming*. 18, 19

PG Programación genética, en inglés *Genetic Programming*. 20

PID Proporcional Integral Derivativo. 32

RCGA Real Coded Genetic Algorithm, Algoritmo genético con cromosomas de coindificación real. 107, 109, 110, 112--117, 119--121, 129, 132, 134, 153, 157, 159--161, 167

SP Tasa de rendimiento, del inglés *Success Performance*. 92, 93

SR Tasa de éxito, del inglés *Success Rate*. 39, 91--93, 103, 105, 109, 111--115, 119, 120, 125, 127, 130, 132, 133, 137, 141, 144, 145, 148, 152, 153, 157, 159, 166

TUB Technische Universität Berlin. 14

UCLA University of California - Los Angeles. 14

UM University of Michigan. 14

UMDA *Univariate Marginal Distribution Algorithm*. 27